University of Deusto

Shades of Internet: Web Security and Privacy Analyses, and Offensive Techniques

Iskander Sanchez-Rola

Bilbao, September 2018

University of Deusto

Shades of Internet: Web Security and Privacy Analyses and Offensive Techniques

Dissertation submitted by
Iskander Sanchez-Rola
for the degree of
doctor of philosophy

Supervised by
Dr. Igor Santos
Dr. Davide Balzarotti

Author | Co-advisor | Co-advisor

Bilbao, September 2018

University of Deusto

Shades of Internet: Web Security and Privacy Analyses, and Offensive Techniques

Dissertation submitted by
Iskander Sanchez-Rola
for the degree of
doctor of philosophy

Examiners
Dr. Leyla Bilge
Dr. Aurelien Francillon

Jury
Dr. William Robertson/Dr. Manuel Egele
Dr. Nick Nikiforakis/Dr. Michalis Polychronakis
Dr. Xabier Ugarte-Pedrero/Dr. Pablo Garaizar

Bilbao, September 2018

function dedicate(recipient) {alert('To my dear '+ recipient)} var name = prompt('Write your name:'); if (name == 'null'){dedicate('reader')}else{dedicate(name)}

Abstract

Since in the origins of the Internet in 1969 (known as ARPANET back then), and the creation of the World Wide Web (WWW) in 1990, much has changed. At the beginning, there were not many security and privacy attacks performed, but nowadays, millions of web attacks are happening all the time. Even if many attacks and vulnerabilities have already been identified, there are many waiting to be discovered or created.

In this thesis, we followed a two path approach, one focusing on the analysis of problems and vulnerabilities, and another based on finding new methods to attack the security and privacy of web users. The first will allow us to understand how widespread are the problems and their possible consequences. The second will allow us to anticipate to possible future malicious attacks.

In the analysis path, we started analyzing the tracking ecosystem on the surface web. As we found that most websites performed some type of tracking, we took a step further analyzing the deep web. After, we analyzed how the most common user/website interaction (a click) could create many security and privacy problems for the user. To finish this path, we checked how the biggest online privacy related regulation has affected the global web tracking ecosystem.

Regarding the offensive attacks path, we discovered three completely new techniques that could vulnerate the security and privacy of a web user. The first one allows to detect the different extensions installed in the browser, bypassing the specified browser extension resources control policies. The seconds allows to fingerprint a computer based on the detection of small imperfections on the clocks of the computer. The last one allows to detect previous accesses to third-party websites simply using some lines of JavaScript.

Acknowledgements

5 0 4 b 0 3 0 4 1 4 0 0 0 2 0 0 0 8 0 0 0 c 6 f 1 8 4 d 0 0 4 3 f 3 2 2
f 2 0 8 0 0 0 0 7 0 2 1 0 0 0 0 0 1 0 0 1 c 0 0 7 8 5 5 5 4 0 9 0 0 0 3
6 8 f 2 7 f 5 b 5 a f 4 7 f 5 b 7 5 7 8 0 b 0 0 0 1 0 4 e 8 0 3 0 0 0 0
0 4 e 8 0 3 0 0 0 0 a 5 5 9 0 d 5 4 5 3 d 7 1 d b f 2 f 7 9 2 1 8 4 0 c
9 1 a f 5 8 4 7 9 5 d 6 1 a 6 4 b 5 3 9 5 4 d 1 7 e c d 1 1 a d b 3 a 1
e 1 7 4 a 2 a 8 a c 1 c 5 b 0 e e 0 2 c c e 3 1 d a a e 9 4 b 4 c b 0 8 6
0 4 5 5 7 8 f 7 2 6 c 6 9 2 9 b 4 a b 5 8 a b d 8 9 e b 3 7 9 5 0 3 e 6
a c b 4 8 2 1 2 0 7 5 9 d 2 b 2 2 8 2 c 0 1 1 0 8 4 c a 9 1 4 6 6 7 0 0
a 9 9 c 4 b c b d 7 c b c 9 7 7 7 d f b b e f e 5 a 3 c 9 c 9 7 9 f 7 b d
f b c f b d f 7 f f f 5 f b 7 f 3 c e 2 0 a 3 1 0 2 e 6 a f 5 8 9 9 b 8 6 a
f 5 6 3 0 0 c 3 3 0 9 0 4 c 7 e 0 1 3 1 0 8 d 6 3 b e e 7 e d 4 c 7 b 1
0 8 f 6 a 3 1 7 f 9 1 a 8 4 0 7 6 3 b 7 b 1 d b 6 2 2 c 1 c 8 8 c 2 3 1
7 1 3 8 4 6 b 4 8 3 5 8 0 0 3 0 8 9 8 b 0 0 b 8 3 f 9 8 4 8 8 c 4 b 8 2
a 4 c 1 b 2 1 0 3 9 4 9 d 0 3 0 1 f 8 8 3 0 b 1 5 8 8 4 8 b 2 5 1 2 1 c
2 7 6 7 f f 4 4 c e 0 3 3 c 5 c 1 2 f 1 e 0 8 a 7 5 4 1 9 1 9 b b 2 a 4 4
b 5 e 8 9 5 a 5 9 7 2 f 8 6 8 f 0 4 3 4 f d 7 b 5 4 6 6 f b e 6 8 5 9 9
a 9 8 f d 6 a a 9 2 c 4 4 b 1 e 0 b e 8 5 3 f 5 5 c 6 2 d 8 b f f d 9 c 3
a 4 4 4 1 e 7 f e 2 c 9 a 7 d 6 f f 7 2 c 3 3 3 9 a e 4 6 7 b 5 6 9 5 b b
6 6 e 4 b f f f 5 f 3 1 9 3 9 3 b 7 e b 3 f 3 a 5 d c 5 d b f f d c 3 6 b
0 5 a f 1 7 e a d e 7 8 7 3 c f 5 b 7 b c b f 6 e d f f f 3 d b e f b c 5 b
f 1 d e f b 9 5 1 f 5 4 5 5 7 f 7 2 e c d 3 e 3 2 7 6 a 4 f 7 e f 6 f 9 e
9 3 3 f 5 0 d 8 d 4 d 5 f 9 c f d b 2 c d d 4 d e 7 1 a e f 3 9 b f 3 f f
e 8 b e d 4 7 3 b 9 b 7 e f 4 a f f 8 0 7 9 7 4 e c d f d 7 a e 8 f 7 f 7
f 6 3 6 2 f a d 6 7 f 6 f d f 9 9 9 9 b 5 f e e f a e 8 3 2 f 0 c 8 8 6 9 d
6 9 1 7 c 9 1 4 2 c 0 4 4 3 8 2 e c 6 a 5 0 e b e 3 0 5 1 a 1 8 3 2 0 1
c 9 7 3 c b 8 2 2 2 8 6 2 d d 2 6 6 9 d 6 2 b 9 1 4 b 5 6 9 6 0 4 4 7
3 d 7 d f 8 6 8 5 d a b e c a 1 c 4 c d 9 6 e 8 e c 5 7 2 f 8 6 2 8 9 6
a e 3 2 2 b a 7 1 d a c 3 9 3 9 f 3 8 d b 1 d 2 8 0 3 8 a 3 1 9 f 3 f 0
3 5 0 0 7 e 2 2 c 6 4 8 e 5 8 9 c 3 8 1 1 a d 8 e d f 1 a 0 4 6 5 4 1 c
0 2 e 2 b 1 5 c 5 0 8 3 b b 8 6 8 0 b a 4 8 c 8 4 1 0 4 f 9 5 5 7 8 9 e
6 6 9 0 9 4 f 4 5 0 c b a 0 1 5 3 9 e 9 1 4 6 e 6 a 7 8 c 6 c b 0 5 7 3
6 e c e 1 e d 2 e 7 c 2 3 c e 4 3 8 7 9 2 4 0 9 7 c 4 c a d f 3 4 8 1 e 2

2 7 2 5 8 0 f 1 c d f 0 a 9 0 8 a a d 9 5 0 0 1 5 8 8 0 2 a 9 2 2 e 7 3
0 c a f 5 5 4 9 4 2 a e a 7 2 f c 5 c 8 f 5 c b 7 5 d 1 4 0 5 f 3 a 2 0 2
4 a e 8 8 9 9 c 1 c 2 8 5 9 3 4 4 a d 7 0 6 d 0 5 0 b e 3 6 d 4 2 9 3 a
f 8 8 c 0 0 7 e 4 2 0 e a 4 a 9 8 2 b b c a d 7 c a a 8 a 7 3 2 e 7 d e f
0 3 e 0 c 1 6 3 0 2 4 5 d 9 4 4 7 0 8 e e c 9 7 9 2 c 0 d 6 3 6 e 1 d f
3 2 8 0 6 a 5 6 0 b 6 4 6 7 c d 2 2 a 6 7 5 5 c 6 c a e 8 6 1 3 c d 7 0
8 e 8 1 f d 7 4 4 7 8 7 6 6 2 9 8 0 6 4 e f 1 c 9 b a 8 5 2 c 8 b c d 3
a 8 7 5 1 a 6 1 e 5 d 9 b 4 a 3 8 3 e 8 3 f 3 4 9 9 4 4 b c a b e 3 8 9
c 8 7 b 2 e 0 0 9 9 7 2 c 5 c 1 e c 8 9 a 6 7 6 a 8 d 7 7 5 5 8 4 a b f
0 9 e 4 5 4 0 f 0 3 c a 2 4 5 6 e 8 7 f 3 1 b 1 3 7 d 7 4 9 2 b 9 5 c 4
c 4 0 c 8 8 a e 2 1 8 b 7 9 9 b a f e c 2 1 0 7 9 8 6 0 0 a a d 0 0 c 0
9 1 e 1 8 4 2 b 2 7 8 3 e 9 c c 9 f 0 0 f b d 5 8 2 6 8 a 5 e 4 3 e 1 d
1 a 2 9 0 7 c f 8 8 e e c 9 2 9 4 b 0 2 e 9 8 5 b 7 a 7 1 5 1 f 8 7 4 0
2 8 6 e 9 d 8 3 b c f 0 9 3 7 1 5 2 4 9 8 c 0 6 0 e 5 0 0 1 7 e 7 5 3 3
4 7 8 f 2 5 d 2 8 b 8 4 5 c 1 9 7 8 4 0 3 7 c 1 2 e 7 d 6 b 4 3 3 e 0 8
1 6 8 8 5 e 3 f d 9 0 2 4 f 2 3 c 0 0 0 e 8 3 2 a c 5 9 6 9 2 4 9 c 0 4
4 5 4 0 f 2 d 2 d 0 a 7 4 6 0 f 0 e 1 8 2 0 2 7 d 6 e 0 0 e 2 b d 3 8 a
2 5 a 1 7 2 3 4 0 5 1 2 1 4 e 4 0 c 8 2 8 d 5 b 0 d 5 c 2 5 2 8 4 0 f 5
c 3 c 8 6 f 0 b 0 0 c 8 e f 6 a 8 2 1 2 3 9 9 1 0 b 6 5 3 5 e f 7 8 0 2 7
3 0 3 a 8 b 5 1 2 0 8 d 9 d b b 1 e 2 e 1 4 f f 7 8 e 6 2 1 1 4 e 1 8 6
7 6 a e b d 0 5 6 0 b 7 2 c f 8 8 d 4 0 8 a 3 5 0 f 3 c c 5 9 2 8 b 3 7
7 6 5 0 a 0 e 9 0 d 9 a 4 2 b c 8 3 2 e 1 3 c e 7 9 4 f 4 c e 1 b 9 d 7
8 8 2 2 4 1 e 4 a 1 3 8 2 2 0 c 2 3 4 2 b 6 6 3 8 6 2 7 6 b 0 1 8 1 a 9
8 0 b 5 2 2 6 f 6 c e 0 a 4 a 1 f 1 5 8 8 f 0 b 2 b d 0 4 e 5 4 e 1 0 4
8 d 7 a 9 f f d 9 2 8 2 1 4 a 5 1 3 6 e 6 8 3 7 1 5 4 2 1 9 8 f 2 b b 3
3 6 a b 7 7 3 f f 0 8 6 5 3 8 8 b f f 9 0 1 f 4 9 c a 5 d 8 e 8 e 8 f 8 a
9 d c f b a 3 5 0 9 5 3 7 a d 6 5 2 1 2 f 5 b 4 c 6 e 0 1 5 1 9 2 8 7 4
1 9 4 c 2 4 4 9 9 1 0 b c 4 2 6 8 2 c 1 7 a 6 4 5 c 6 2 f 0 7 8 3 4 a 9
a 3 9 f 6 0 1 6 1 8 6 5 1 c c c 6 f e 2 b 6 c 4 a 4 3 0 0 e 9 b c a 2 7 f
1 2 a 0 a e d 0 1 6 6 d e a 5 4 d 7 a 9 a 7 1 2 a 6 3 7 9 a 0 9 6 0 c a
e f a f 3 4 9 f 9 a 2 c 3 1 9 7 2 f d 4 f 5 7 5 1 2 c 0 5 2 6 b c c d d b
8 5 0 b f e 5 a 2 e 1 7 7 4 7 e e d 3 0 d 9 a 8 6 9 6 5 a c 6 d b 0 c 0
d 3 6 7 5 f 7 9 a b 4 4 f 2 6 3 2 c 3 1 a 6 2 f b 2 b 3 b 3 2 c 2 d a f 3
a 7 5 4 c b c c 9 7 e f 1 c 3 c b 2 f f 2 6 0 1 4 e 2 7 b 4 c 4 6 5 f c b
0 a d a 9 2 5 a 4 5 8 1 4 4 9 0 0 c 9 a 7 d 6 a c b 9 3 8 0 2 b 4 6 a 0
d 2 3 a 7 8 c 7 2 f 5 4 8 c d 9 1 8 6 c 2 3 2 f b c 2 a 2 0 b e a 3 3 d
7 8 d 0 4 c 0 2 7 2 d 7 d 9 3 b 5 2 3 5 7 d 3 b f 6 e b c 7 b e 2 a b 6
3 b b 6 3 e f 5 7 e f c c 8 c d 4 2 7 5 b c f 8 7 e d 9 b 6 c 8 9 d 6 2 7
f 5 1 6 2 5 e 8 d 7 3 5 5 c 6 e 4 e 7 9 e 9 c 4 e e e a a 1 6 3 a 3 f 9 5

6 e d e d e f 7 0 c d e a 8 3 c 7 f f 3 2 5 c 5 6 7 f 6 1 b 3 7 c 7 9 4 d
5 8 e f 7 7 3 1 8 6 d 6 0 d 0 4 c 8 2 6 c 0 b e 8 f 6 5 f 7 3 e 4 8 5 a a
6 a b 1 d 3 b 7 4 5 8 6 a 4 a 1 8 5 c 7 e 2 e 6 9 b e 4 5 3 a 7 9 6 d 3
a 5 1 6 3 3 1 9 6 3 7 6 f b 5 8 c 7 7 7 e b c c 2 5 a 1 2 d d 3 c 3 4 a
f a b 2 0 6 1 b 1 2 e a d a d 7 a f 8 2 d f b e 2 a b b a b 2 2 7 8 f 2 b
2 2 6 c 2 9 9 7 6 d 8 b 7 2 6 f e d b a d b 1 3 b 1 5 d 9 3 5 4 f 2 3 7
4 d 6 d 7 1 e 1 a c b a 3 1 6 c f 2 8 7 b c b 9 6 d 3 7 8 6 e b e 6 5 b
4 b 5 6 1 3 6 0 4 a 7 d d 2 f 6 f 0 b b 6 3 b b f 5 8 9 1 7 f 5 0 b c 6
e e b d f 1 9 a 7 a 6 f d d 4 4 6 1 f 5 1 2 5 d d 3 b 9 a 4 c 7 b b 6 c
c b 4 f e a d 7 5 8 5 4 7 f b d 7 e 6 a c 7 c 0 e b 5 5 b 5 d 6 4 b 1 d
5 f 2 a 4 e 7 d 3 e f 4 c 2 8 d 4 7 7 7 b d 5 9 3 0 5 9 d d 7 b 2 8 a 1
3 8 a 9 4 a 0 8 3 e 0 2 c 7 1 5 0 4 7 0 d 9 b 7 a b 7 f b f 7 1 a 0 6 f
a 4 d 0 1 6 6 9 b 0 c f 0 b b 6 1 a e 6 b 2 8 a a e 0 c 3 d 3 0 9 d d 0
6 6 e 8 f f 7 0 3 4 d 3 1 6 3 e b b e 9 5 e 9 5 3 e 7 1 b a 2 c a c 6 c 3
8 a 2 e 8 0 1 d d 8 b 9 9 0 7 3 3 d 2 a 7 2 a d 5 1 d b b 8 7 f 1 a b b
5 a 6 4 c 9 9 d 5 f 8 7 c a 9 5 d 1 d b 4 2 1 e f b 3 e 4 f 4 9 4 f 7 9 f
4 8 d f d a 6 d e f c e 1 b f d f b b 3 5 b c 4 3 d c 6 a c 9 b 7 b 3 3 4
3 9 b 0 d a 4 9 6 c 7 0 9 b 0 6 7 7 f c 3 c 6 f b 7 5 3 1 6 d 4 3 a a 1
e 9 5 7 e 1 1 a 9 d 9 7 6 a b 2 1 b f 6 0 5 4 1 5 f 9 a b 5 e e 5 9 e 5
6 7 5 0 4 0 1 7 2 a 6 9 e 6 b 3 6 9 8 7 5 0 3 f 6 a 1 9 d c f 2 d e 8 f
8 d 6 1 c 1 b a a 1 9 0 d 5 d 9 3 3 d 9 c 7 6 6 0 c 3 a 5 b 1 e 6 4 3 3
b 4 a 1 f a d 8 9 2 d 2 7 0 0 6 8 7 9 c 4 a 6 6 a 2 2 1 7 4 c 3 1 e 6 0
c f 7 7 7 7 6 2 e 1 6 1 f 1 8 6 b e a 3 4 0 c a 6 6 5 5 a 0 7 e 7 5 6 1 5
7 d 9 3 1 9 b 4 7 7 6 e c 0 f a 6 1 b b 3 a f 6 4 e f 9 e 0 f a b 1 1 7 8
e 9 8 0 c 8 b 7 2 6 7 7 b 6 f 8 5 6 a d 6 7 5 d b 2 5 c 3 a d 4 7 a d f
f 9 9 6 d 7 f f 9 7 c 5 e 4 e 5 6 5 c f 9 5 1 5 d f 6 8 c f d 8 2 7 e e b
c e a 5 e 0 d e 3 6 0 2 9 4 a 6 6 c d 7 9 5 c f 2 4 9 3 8 3 b 7 e d 8 3
8 6 b a f c a 9 6 f 4 7 7 f 1 e b 6 9 f 0 0 4 d 3 7 3 b 0 f c 6 9 8 d 3 a
7 6 2 2 c 7 3 3 3 6 7 8 d 8 d 9 5 d a f c 8 1 f 1 9 9 9 7 2 d e 5 a 6 3
c a 9 f 9 5 a 6 5 6 9 7 3 6 2 c e 1 f 9 7 c 7 1 d 3 0 c c f d 4 2 b 7 c f
f 4 e 2 2 3 a 7 8 d 9 6 c 6 8 f b 2 c e f 6 d 5 b 4 4 c 2 f 3 b 4 0 8 0 3
3 c 7 e b b b f 2 5 7 6 b 8 0 b d e a 7 b 1 f 9 5 2 5 1 4 1 9 e 1 6 a d
4 8 f 3 2 1 0 5 5 a 5 3 0 3 6 6 2 5 b 9 5 6 b 6 5 f a 2 9 7 4 c b 2 9 0
8 5 c 8 5 7 5 0 1 1 3 a a 5 c a 9 e 9 8 4 3 b 3 6 f a 6 a a 6 f 0 2 9 5 8
f b 5 0 7 d 1 e f 4 a d 5 0 2 a e 7 1 6 5 9 0 f 1 7 f d b 8 a 8 4 f c 1 b
4 d c c f 0 4 1 a 1 1 f a b 8 7 4 5 0 d d c 2 8 d 8 0 6 8 d 8 a 6 1 7 6
3 4 4 7 4 7 a 0 8 0 7 6 4 9 e 5 0 a 9 6 2 6 b c 6 4 d 4 7 7 1 e 9 a e 7
4 1 0 5 5 8 a d 6 5 3 1 d c b 8 5 a b 2 5 6 8 4 9 3 6 6 b a 8 9 3 2 3 a
2 a 7 5 0 9 c 8 7 1 a 5 e 5 1 c c 4 e 1 b 8 f 7 a 1 8 f 3 e e 3 2 2 5 f 0

0 c 0 2 9 5 1 c f 8 5 f a 4 2 8 1 1 1 6 4 7 6 5 5 6 0 3 9 c 1 6 0 f c 3 3
d 1 9 6 e 2 e d 0 4 1 a 2 d 4 c 7 0 d 1 0 0 1 8 5 0 c 0 5 d 5 f 3 4 1 f
4 8 9 1 8 a e e 9 7 b d 7 4 2 a 0 1 c 6 1 f 8 3 2 8 7 7 9 9 e 5 0 2 6 6
2 f 1 c 4 5 f 6 b f 3 7 6 2 5 a 0 b 6 1 a 9 4 4 2 e 7 0 6 5 2 3 1 f 5 2 0
a a 7 1 1 5 9 d 3 c a 7 a 2 4 1 8 5 d 2 7 f 0 5 6 4 5 9 a 1 4 8 1 4 8 7
9 1 9 4 4 7 9 c 5 4 a b 8 b d 9 8 1 6 0 c 1 a a 7 0 7 1 0 8 8 9 1 6 a 5
0 5 1 6 d 2 3 3 5 8 4 6 0 f 0 5 1 0 c e e d 0 b 3 e 2 7 b 9 3 e 4 3 0 4
f b 7 4 9 4 b 3 a 2 b c 5 9 c 9 d 3 8 9 f 1 5 a 2 6 7 1 4 2 0 5 2 f 2 4
c 8 9 0 e 8 c 9 3 c 1 4 e 3 b d 8 1 d 2 2 b b e b 0 e 5 e 5 7 3 b 4 f 2
e c 8 5 8 8 4 4 7 4 b f 4 b 0 0 1 0 b 4 7 e f 5 d e 7 9 7 c d 8 e 7 8 e
3 4 5 c 6 0 7 b 6 f 2 3 d 3 7 6 8 e f b f 3 7 a c 1 e 7 2 c 5 b 4 b 3 5 8
e 2 4 7 0 d f 5 8 e 2 2 e 2 8 a 3 0 4 6 b 2 b 6 f c 5 2 9 2 b 1 8 0 7 d
c 1 0 f 1 b 9 a f c 2 4 e c c 5 0 4 1 9 e d 7 c 5 3 c 0 9 8 8 a 7 0 b c b
e a e 1 3 1 5 e 5 a 7 9 4 a 6 e 2 8 1 1 4 9 1 5 f 9 9 c 5 8 c 1 b d 5 e
b 6 a 7 7 3 5 f 3 1 f 9 d 6 9 0 6 6 0 b c c 7 b 7 f 2 e e 0 9 e 9 a d 6 f
d 0 2 c a e f 3 f f a f 3 5 a c 0 7 5 7 e c 6 3 f 5 c 5 9 1 d f 4 9 6 d 2
6 d d e 2 1 d 4 5 9 a 0 9 4 7 a 9 1 1 0 e 8 4 4 b 1 a 2 9 4 c c d 7 7 6
3 3 1 b e c 9 8 1 e c 1 c 2 5 b 0 f a 6 7 2 2 8 1 8 5 b 7 1 d f 9 0 c a
7 c 0 b 1 a c c 9 7 a e f e b 4 5 6 f c 6 9 6 5 a 3 8 e c 3 0 0 8 0 5 8 a
2 3 f e 4 f f 5 0 4 b 0 1 0 2 1 e 0 3 1 4 0 0 0 2 0 0 0 8 0 0 0 c 6 f 1 8
4 d 0 0 4 3 f 3 2 2 f 2 0 8 0 0 0 0 7 0 2 1 0 0 0 0 0 1 0 0 1 8 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 b 4 8 1 0 0 0 0 0 0 0 0 7 8 5 5 5 4 0 5 0
0 0 3 6 8 f 2 7 f 5 b 7 5 7 8 0 b 0 0 0 1 0 4 e 8 0 3 0 0 0 0 0 4 e 8 0
3 0 0 0 0 5 0 4 b 0 5 0 6 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 4 7 0 0 0 0
0 0 2 d 0 9 0 0 0 0 0 0 0 0

Contributions

Background & Motivation

[C.9] *Iskander Sanchez-Rola*, Xabier Ugarte-Pedrero, Igor Santos and Pablo G. Bringas. Tracking Users Like There is No Tomorrow: Privacy on the Current Internet.

[C.8] *Iskander Sanchez-Rola*, Xabier Ugarte-Pedrero, Igor Santos and Pablo G. Bringas. The Web is Watching You: A Comprehensive Review of Web Tracking Techniques and Countermeasures.

Web Security and Privacy Analyses

[C.7] *Iskander Sanchez-Rola* and Igor Santos. Knockin' on Trackers' Door: Large-Scale Automatic Analysis of Web Tracking.

[C.6] *Iskander Sanchez-Rola*, Davide Balzarotti and Igor Santos. The Onions Have Eyes: A Comprehensive Structure and Privacy Analysis of Tor Hidden Services.

[C.5] *Iskander Sanchez-Rola*, Davide Balzarotti, Christopher Kruegel, Giovanni Vigna and Igor Santos. Dirty Clicks: a Study of the Usability and Security Implications of Click-related Behaviors on the Web.

[C.4] *Iskander Sanchez-Rola*, Matteo Dell'Amico, Platon Kotzias, Davide Balzarotti, Leyla Bilge, Pierre-Antoine Vervier and Igor Santos. Can I Opt Out Yet? GDPR and the Global Illusion of Cookie Control.

Offensive Web Security and Privacy Techniques

[C.3] *Iskander Sanchez-Rola*, Igor Santos and Davide Balzarotti. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies.

[C.2] *Iskander Sanchez-Rola*, Igor Santos and Davide Balzarotti. Clock Around the Clock: Time-Based Device Fingerprinting.

[C.1] *Iskander Sanchez-Rola*, Davide Balzarotti and Igor Santos. Baking-Timer: Privacy Analysis of Server-Side Request Processing Time.

Contents

Figure Index

Table Index

Part I

Background & Motivation

"I'm not locked up in here with YOU.
You're locked up in here with ME."

Alan Moore (1953–)

Chapter

1

Introduction

Cambridge Dictionary [73] defines privacy as "someone's right to keep their personal matters and relationships secret". Until the beginning of the digital age, the meaning of privacy was pretty clear for the majority of the people, and was regularly respected by most companies and services. However, everything started to change with the Internet. After the initial years, companies started to realize that tracking users online and attacking their privacy could end up in a very profitable business.

Nowadays, web tracking is a widespread technique on the Internet that gathers user data to perform online advertisement, content personalization, or user authentication. In general, web tracking allows third-party or first-party websites to know the users' browsing history and browsing configuration to these ends. These techniques can be used to improve users' experience and to enhance their browsing on the Internet. However, since they sometimes involve retrieving user data, even without their consent, they can be considered a privacy violation by itself in some occasions. Web advertising companies refuse to acknowledge web tracking as a threat and they even publicly defend web tracking on the basis of its relevance in the Internet economy [254]. Web-tracking techniques can be of stateful or stateless nature, depending on whether or not they require data to be stored in user's computer to properly function. It is usually considered that stateless tracking methods are harder to limit and block because they

easily bypass common countermeasures against tracking such as private browsing or removing cookies [14].

Recent work [14, 186, 235, 215, 13] has studied the prevalence of different types of web tracking, fingerprinting and privacy violations. These works have found that web tracking is a very usual practice and that not only it is based on its less damaging forms as web analytics but also advanced fingerprinting techniques such as font probing or canvas/WebGL fingerprinting.

1.1 Hypothesis and objectives

Based on the problem described in this chapter, we formulate the following hypothesis to test during this dissertation:

> *"Web security and privacy vulnerations are a worryingly common and known problem in the current Internet. Nevertheless, new offensive techniques and analyses can be performed."*

Considering the presented hypothesis, we define the general objective of this dissertation:

General objective. *Improve web security and privacy from different perspectives.*

This objective can be divided into the following specific objectives:

Specific objective 1. *Conduct comprehensive analyses.*

Specific objective 2. *Create new offensive techniques.*

Given these objectives, we define the following operational objectives that will guide the research during this dissertation.

Comprehensive analyses

Operational objective 1. *Detect a security and privacy problem on the web that has never been analyzed, and proposed an analysis method to know how widespread is it.*

Operational objective 2. *Develop a tool capable of obtaining all the information needed, and perform a comprehensive analysis of the problem.*

New offensive technique

Operational objective 3. *Perform an in-depth analysis of possible methods that could be used by malicious actors, executing JavaScript in the browser.*

Operational objective 4. *Develop a model of the attacks and perform multiple tests to understand the potential implications for the users.*

1.2 Research methodology

In this section we will explain the methodology followed during this thesis and each of the contributions. The methodology consists in multiple stages that happen in a cycle, allowing to refine the methods proposed.

1 **Identification of open problems.** The first step is to identify possible open problem of the research area, performing an initial relevant publications analysis. Even if the analysis is not deep, it should be as large as possible, to try to get, at least, the general idea of the topic. This will help detecting specific problems that have not been previously addressed by the community.

2 **Literature review.** Once certain open problems have been identified, the seconds step is to perform an exhaustive analysis of the publications that are more concretely related to those topics. However, not only contributions should be analyzed, surveys and not scientific technical reports can give a really interesting and helpful view, both to center the idea or to understand it better. One of the key point when reading this publications, is to always have a critical interpretation, that could allow to detect limitations or weaknesses of the presented approaches.

3 **Hypothesis and objectives.** This phase is a crucial step that will determine the following steps. After we identified the open problems and deeply reviewed the literature, we have to define the hypothesis and the specific objectives that will guide all the future developments of the thesis or contribution. Finally, we should prepare a work plan, indicating possible deadlines and milestones.

4 **Method development.** Once we have defined the work plan, we start with the development. Even if we have some specific milestones, this

is a dynamic process, so something small modifications and updates should be performed. Moreover, this step includes the study and improvement of our technical skills to obtain the best possible outcome.

5 **Evaluation and hypothesis confirmation/rejection.** After the development finishes, the next step is to design and discuss the different possible experiments to perform in order to confirm or reject the proposed hypothesis. Once we think that the evaluation is adequate to test the it, we start executing the experiments. If obtained result support the proposed hypothesis, we will confirm it. Otherwise, we will need to redesign our approach or even reformulate our initial hypothesis.

6 **Results presentation/publication.** The last step is to present the obtained results to the scientific community, in order to get valuable feedback that could help to improve the idea. Sometimes, we will need to perform additional analyses or experiments.

1.3 Structure of the document

The rest of this document is structured as follows. Chapter 2 introduces the different concepts related to web privacy and tracking, explaining existing types and applications, regulations, and countermeasures. Then, the contributions of this thesis are divided in two different parts. Part I is based on comprehensive analyses of different open problems of the web security and privacy community. Chapter 3 presents the first generic large-scale analysis of different known and unknown web tracking scripts on the Internet to understand its current ecosystem and their behavior. Next, in Chapter 4, we developed a dedicated analysis platform and used it to crawl and analyze over 1.5M URLs hosted in 7257 onion domains. We performed the largest structure analysis and we measured for the first time the prevalence and nature of web tracking in Tor hidden services. In Chapter 5, we implemented a crawler that performed nearly 2.5M clicks on 118K pages looking for signs of misbehavior. We analyzed all the iterations created as a result of those clicks, and discovered that the vast majority of domains are putting their users at risk by either deceiving the real target of links or by not providing sufficient information for users to take an informed decision on their actions. In the last chapter of the first part (Chapter 6), we performed an evaluation of the tracking performed, after the GDPR entered

into effect, in 2,000 high-traffic websites, different by categories and hosted both inside and outside of the European Union.

Part II is centered on creating new offensive techniques that could be used to attacks the security and privacy of web users, in order to avoid future possible malicious attackers. In Chapter 7, we present a timing side-channel attack against the access control settings and an attack that takes advantage of poor programming practice, affecting a large number of Safari extensions. Due to the harmful nature of our findings, we also discuss possible countermeasures against our own attacks and reported our findings and countermeasures to the different actors involved. Next, in Chapter 8, we show a new way to compute a hardware finger- printing, based on timing the execution of sequences of instructions readily available in API functions. Due to its simplicity, this method can also be performed remotely by simply timing few seemingly innocuous lines of JavaScript code. In the last chapter of this part (Chapter 9),we propose a new history sniffing technique based on analyzing the server-side request processing schema through web timing that only uses first-party cookies, considered harmless even by web browsers, and usually not employed for web tracking. Finally, Chapter 10 summarizes the conclusions of the thesis.

"If you want to keep a secret, you must also hide it from yourself"

George Orwell (1903–1950)

Chapter 2

The Web is Watching You: A Comprehensive Review of Web Tracking Techniques and Countermeasures

In this chapter, we comprehensively review the web tracking research area, extending our previous review [245]. We study this topic from a web security research perspective, describing both web tracking techniques and defenses. The goal is to bring understanding of the current state of the art in web tracking to allow a better understanding of its landscape and the proper discussion of its implications and future research trends. This review, is to the best of our knowledge the first one that reviews multiple web tracking techniques, countermeasures and legislation.

2.1 Web Tracking Applications

Albeit some web tracking techniques are privacy-invasive and raise a serious concern for users' privacy, they are an extended practice in the Internet.

In this section, we review the different applications of tracking in the web and we detail non-privacy-invasive alternatives that can be used for these applications.

2.1.1 Types of Web Tracking

Roesner et al. [235] defined a taxonomy of third-party cookie-based web tracking that is currently the most common form of tracking. For this, they focused on two aspects: the functionality of the tracker and the type of behavior. Regarding the functionality, the following categories were defined: (i) third-party advertisement, (ii)third-party advertisement with pop ups, (iii)third-party advertisement networks, and (iv) third-party social widgets. With regards to the type of behaviour, the authors defined defined the following ones:

- **Behavior A (Analytics).** The tracker serves an analytics engine for each website. Only tracks users within that specific site
- **Behavior B (Vanilla)**.The tracker exhibits third-party storage that can be get and set only from a third-party position to track users across sites.
- **Behavior C (Forced)**. The cross-site tracker forces users to visit its domain directly (such as popups or redirections), placing it in a first-party position.
- **Behavior D (Referred).** The tracker relies on a type B, C, or E tracker to leak unique identifiers, rather than on its own client, to track users across sites.
- **Behavior E (Personal)**. The cross-site tracker is visited by the user directly in other contexts.

These last categories are not mutually exclusive with the exception of B and E, because the user can only visit the web tracker's domain. In their study in 2012, they found out that in the top visited 500 webpages in the well-known Alexa ranking, the most common tracker type was the B (Vanilla) one. With these categories, we can understand the different levels of danger that web trackers pose depending on their use of data. In addition to these categories, advanced fingerprinting is an stateless tracking technique that bypasses user's browsing configuration as we will be detail later.

2.1.2 Advertising and Analytics Services

The most well-known and extended use of web tracking and arguably the main application of it, are web advertisement and analytics. Web tracking is used in analytics to track the visitors of a particular website, its articles and sub-pages. In addition other information and demographics such as browser version or operating system can also be obtained [19]. These user profiles are stored and updated as the user interacts with websites. In advertisement, the browsing history of a particular user is retrieved by gathering the user identifier from the websites that host the particular advertisement network, knowing which websites she has visited. Some of the aforementioned techniques can be used to enhance user identification and allow to share the browsing history among different advertisement companies. Based upon the user profiling, the advertisement network will also update its advertisement placement specially targeting to the user visiting the website in its network.

Although these implementations may variate, nearly every vendor has adopted one of two typical models. Some offer analytics as a paid service and they cannot utilize the client's analytics information, protecting the obtained information. Other vendors offer a free analytics service, but they use the obtained data for ad targeting, market understanding, and so on. Advertising companies do not always depend on the data sold by the analytics services. They use their own techniques to categorize the user. Information transference between banners is one of the most employed techniques [235]. In addition to pre-packaged solutions, some websites use their own implementations [141]. They are sometimes even obfuscated to evade detection systems. As these methods do not follow any specific information flow, they are much more difficult to detect and stop.

2.1.3 Alternatives to Privacy-Invasive Tracking

Various advertising companies have declared that web tracking is required to maintain the web economy [254]. However, other techniques have been proposed for analytics and targeting preserving users' privacy. *Adnostic* [272] was proposed by Toubiana et al. to overcome the privacy issues of online behavioral advertisement. They proposed a practical architecture to performing the user targeting directly in the user browser instead of sending any information to third parties. *PrivAd* [126] implements a very similar approach but utilizes a trusted party to anonymize the client. *RePriv*

[109] uses the browser to allow interest profiling and generically enables personalization. Bilenko et al. presented a technique [34] to store the user profile and recent browsing history in a cookie without the intervention of any third-party. *ObliviAd* [27] is an approach for privacy-preserving online behavioral advertisement that employs secure hardware-based private information gathering for distribution of advertisements. All these approaches can provide a continuation to web advertisement without violating users' privacy.

2.2 Web Tracking Techniques

2.2.1 Stateful Tracking

This type of web tracking techniques use the available methods to store information within the client's computer. To this end, third-party websites access different aspects of the websites to retrieve user data and to store it.

Lou Montulli proposed *cookies* mainly to allow users' stateful web navigation [247]. Cookie usage permits a website to store data on the users' computers that is retrieved when the user returns to the site. In this way, websites can maintain the navigation state that otherwise would be lost and thus, increase the usability.

Although they were not designed for it, the abuse of their stateful nature started shortly after their first appearance. Since websites are composed of different resources that may be allocated in the web hosting, the main page and also in third-party servers, these external resource providers have the capability of including cookies along with their provided resource. If the third-party server provides resources to a large number of other websites that this particular user visits, it can certainly gather user's browsing history and profile her browsing habits. For example, a website `site.com` includes an image from a third-party domain called `advertisement.com`. The server `advertisement.com` sends the image along with a `HTTP Set-Cookie` header, that will be stored on her machine. The user visits another website `site-two.com` that also utilizes images from `advertisement.com`, when asking for the image `site-two.com`. It will send user's previously set cookie to `advertisement.com`. It will recognize the user and start tracking her. This behavior is called *third-party cookies* and it is arguably the most extended technique for web tracking.

The functionality of HTTP cookies was later adopted by other components of the browser like Flash Cookies [26] (also named Local Shared Ob-

jects or LSO), HTML5 [284], or JavaScript. Flash cookies store more data, they lack an expiration date, and they are not controlled by the browser. HTML5 allows websites and third-party trackers to store information in the browser without setting a cookie. The `Window.name` is a non-persistent property of JavaScript that can be used to share data between different websites.

Third-party cookies raised the concern of the research community [235, 215, 159, 157, 84] that was also extended to popular media and the general public. As a result, the community responded in several ways. For example, comScore (a popular online advertisement company) presented a study [60] that showed that 1 out of 3 users removed first and third party cookies within a month. Extensions also appeared to block third-party tracking [57, 80] and to visualize cookie sharing [25]. Finally, the currently widespread *Private Browsing mode* was born to allow users to navigate the web without leaving any trace in their local storage.

However, trackers responded with a set of techniques derivatively created to circumvent these solutions. Cookie syncing is a practice of tracking companies to bypass the Same-Origin Policy and allow different trackers to use the same user identifiers and share them. This technique synchronizes cookies among trackers by passing user identifications. Since each website cannot read other cookies, cookie syncing or synchronization provides a method to, according to Google, facilitate targeting and real-time bidding [235]. *Cookie respawning* and *Evercookies* intentionally abuse the storage methods of the browsers to restore the previously removed cookies. Soltani et al. [257] discovered the use of Flash cookies to regenerate removed HTTP cookies. In a later study [26], they discovered several websites using *ETags* and HTML `localStorage` API to respawn cookies. Evercookie was created by Kamkar [147] as a resistant tracking method with different storage mechanisms such as Flash cookies, `localStorage`, `sessionStorage` and *Etags*. Evercookie also employed various novel methods, working altogether to regenerate cookies.

2.2.2 **Stateless Tracking**

Stateless tracking does not require to store any information on the users' computer. Stateless device fingerprinting has become an increasingly common practice performed by advertisement and anti-fraud enterprises. This stateless techniques permit these companies to bypass the private browsing mode and also the current cookie-related regulations in Europe and the

United States. In addition, these techniques allow advertisers to increase the previous gathered user data and an easier sharing of the user identifications between different tracking services. Currently, the are several features to uniquely fingerprint a device that can be gathered from different aspects of the browser such as *Javascript* or *plugins*.

JavaScript-based device fingerprinting is performed by inspecting its accessible browser resources. For example, navigator contains data about the browser vendor and version, supported plugins and MIME types, and information about the operating system and architecture. Another object commonly used is screen that contains information about the user monitor resolutions as well as the color and pixel depth. Mayer [185] experimented fingerprinting 1,328 users by hashing the contents of the JavaScript accessible browser features `navigator`, `screen`, `navigator.plugins`, and `navigator.MimeTypes`, allowing to uniquely fingerprint more than 96% of the users. Eckersley [79] extended that browser features by adding a list of the installed fonts, timezones and a browser's ACCEPT headers, combining them to create an unique device-specific identifier. To this end, *Panopticlick* [81] was developed and evaluated on around half million users, identifying correctly the 94.2% of the half million users, demonstrating that it was possible to be identified and tracked without the need of any stateful client-storage mechanism. Eckersley also demonstrated that the list of installed fonts was the most accurate feature for device fingerprinting. In addition, the browsing history can also be gathered by abusing the JavaScript¡s visited-link color feature [198]. Other researchers have proposed the usage of performance benchmarks to identify the different JavaScript engines [194], raising errors of the standard tests [206] or computing the differences by elements created with the `canvas` HTML element [199].

This last technique is known as *canvas fingerprinting* and by the utilization of the Canvas API of modern web browsers, subtle differences in rendering the same text or WebGL scenes can be abused to extract a unique fingerprint. The main property that allows this technique to work so well is the fact that the same text can be rendered in different manners depending on the operating system, due to the differences in the rasterization such as anti-aliasing, hinting, sub-pixel smoothing, system fonts, API implementations or the display. The technique draws as many different letters as possible to the canvas with the intention of maximizing the diversity.

2.3 Web Tracking Analysis and Detection

The first techniques that were adopted to block web tracking were based on blacklists such as *EasyPrivacy* [15] and *Ghostery* [57]. These solutions contain a list of names of scripts known to perform some sort of web tracking and domains known to host web tracking. In this way, when a website and the URL of its first-party or third-party scripts is available, they are searched in the blacklists. If a script in the website is named as one in the blacklists or it is hosted in a blacklisted domain then its functionality is blocked by the blacklisting solution. Other solutions exists like *Privacy Badger* [80] or *Disconnect* [76] that utilize heuristics to determine that a third-party domain is loading and sending content to the website.

In addition to these solutions, several researchers have performed studies of web tracking. In this way, one the first studies analyzed the prevalence of HTML cookies on the Internet [158]. Mayer & Mitchell [186] analyzed the diverse web tracking methods, developing a framework to evaluate websites' privacy. Roesner et al. [235] developed a taxonomy to categorize the different web tracking methods, measuring their prevalence. Nikiforakis et al. [215] focused their study in three well-known fingerprinting companies, discovering that 40 websites out of 10,000 sites used advanced fingerprinting techniques such as font probing. Acar et al. [14] presented *FPDetective* that based on different manually derived rules, was capable of detecting fingerprinting and advanced web tracking techniques. Later, Acar et al. [13] focused on the recent canvas fingerprinting method and, by the means of manually-set rules, measured its prevalence in the Internet, finding that 5% of the websites in the top 100,000 of the Alexa ranking used this type of fingerprinting. They also measured the prevalence of cookie syncing and rewspaning techniques, showing that they are extended practices.

2.4 Countermeasures to Web Tracking

2.4.1 Anti-tracking Techniques

Several browser configurations can be used in order to avoid some specific types of web tracking. Even though the most effective countermeasure is to just disable JavaScript since most of the attacks described use or depend somehow in it, it is not desirable since it affects common user browsing. A more viable option is to use the temporary modes such as private or guest

mode that most current browsers implement. In this way, the browser will not save or cache any visited website or downloaded file [17] and will avoid classic types of cookies. Since most fingerprinting techniques retrieve specific configurations of the browsers in order to compute a unique identification for the user, another possibility for avoiding being tracked and fingerprinted is to disable certain specific font sets, disabling cookies and using domain and script blacklisting techniques. Anyhow, these countermeasures will not avoid every type of web tracking technique and advanced techniques may bypass these blocking methods. Unfortunately, disabling JavaScript would prevent some methods for web tracking but it also causes many websites to render incorrectly. Disabling other secondary features used in web tracking is a more promising approach because the number of websites that rely on them is smaller.

Spoofing a user profile can also be considered as a possible countermeasure against web tracking. However, the best possible spoofing configuration to avoid web tracking will require every user to share the same user profile, which renders inviable nowadays. Spoofing data and using different random profiles would help to circumvent web tracking because it hinders the uniqueness of the user's browser. Some of the properties to spoof are browser, platform, time zone or screen resolution. Nevertheless, spoofing could be counterproductive because these attempts to hide the identity can also be used for device fingerprinting [215].

There also exist fully functional anti-tracking web browsers (e.g., FlowFox [67], *TrackingFree* [222], or Privaricator [214]) that implement a precise and general information analysis and control, devoted to protect the user against web tracking. In order to limit their effect in user browsing, these privacy-aware browsers seek a very low performance overhead. In addition, there is also some work focused in the analysis of user's browsing [129]. In this way, all the accessed websites could be analyzed without exception, taking into account that the user is the weakest link in the security chain. By applying taint analysis or dynamic controls, and using several policies, it is possible to detect web tracking [251, 54]. The main problem of these methods is that they only take into account certain fields and privacy attacks. Understanding and controlling every type of privacy attack would enormously improve web-browsers. Nevertheless, the biggest disadvantage of a general control method as taint analysis, is its computational complexity.

2.4.2 Standardization

One possible solution to the problem of web tracking is to standardize the control of the information that is being transmitted. Two main projects have been advanced for giving users control over their personal data: *Do Not Track* (DNT) [184] and *Platform for Privacy Preferences* (P3P) [290].

Do Not Track is a proposal that combines technology and policies in order to declare user's preferences regarding web tracking. This information is sent via an HTTP header: `DNT`. All modern browsers (e.g., Chrome, Firefox, Opera, Safari, and Internet Explorer) support a Do Not Track opt-out preference (i.e., `DNT: 1 header`). This policy also indicates that websites must stop tracking the user for whatever reason when they receive a `DNT` header.

Platform for Privacy Preferences is devoted to facilitate to websites the task of communicating their privacy habits in a standard format that can be automatically obtained and understood by user agents. Users have the possibility of coming to a decision based on the privacy practices indicated by the website [44]. Thanks to that, users do not need to read the privacy policies of all the webpages they access, they just need to read it's practices. Websites implementing these policies have to make their habits public. Browsers can help the user to interpret those privacy habits with user-friendly interfaces.

Although many stakeholders (policy makers, consumer advocates and researchers) think that Do Not Track could decidedly reduce tracking and data collection on the web, as the final decision of taking it into account only resides in websites, it is not followed as expected [183]. The case of P3P is similar, due to the lack of support from current browsers for the implementation, the P3P Specification Working Group suspended the project.

2.4.3 Regulations

After understanding the magnitude of the problem, we should understand the existing regulations in the United States and European Union [186]. It was not until recently that these regulations were introduced in order to restrict large-scale collection of personal data [115].

In the *United States*, one of the missions of the Federal Trade Commission (FTC) is the promotion of consumer protection. They can only prevent practices of businesses that are either unfair or deceptive under 15 U.S.C. § 45. First violations will incur on a small payment, but subsequent

violations get big monetary penalties. On 2012 the FTC issued its final report [90] establishing four best practices for companies to protect the privacy of all American consumers and give them the possibility to have more control of tracking options and personal information collection. The report expands on a preliminary report released in 2010 [89] which proposed a framework for consumer privacy control because of the new technologies that allow information collection that is often not perceivable by consumers. The objective is to balance the personal data of consumers with innovation.

Regarding the *European Union*, the Directive 2002/58/EC on Privacy and Electronic Communications, also known as E-Privacy Directive, indicates that the use of electronic communications networks to store information or to gain access to information stored in the terminal equipment of a user is only allowed on condition that the user concerned is provided with clear and comprehensive information about the purposes of the processing, and is offered the right to refuse such processing [85]. If the above indications are not met, penalties could be up to 2% of the revenue. The Article 29 Working Party (WP29) addresses the topic of device fingerprinting in the Opinion 9/2014 , which extends over the previous Opinion on Cookie Consent Exemption [23], and indicates that websites cannot process device fingerprints which are generated through the gaining of access to or the storing of information on the user's terminal device if there is not a explicit consent of the user (unless some specific exemptions) [24]. In May 2018 the General Privacy Data Regulation (GDPR) was introduced, that enforces strict limitations on the handling of users' personal data and, in particular, on tracking their activity on the Web [10].

There have also been attemps of *self-regulation*. In 2009, many of the largest advertising and marketing companies and associations, supported by the Council of Better Business Bureaus, created a self-regulatory program with the principal objective of giving total control over the collection and use of private data to the users [74]. Websites should have clear options regarding to the data collection and use, letting the user decide if they want that collection or not. There should also be a limit on the specific data type obtained if it is sensitive information. Until that moment, all the different actors worked interdependently in this area. Nevertheless, it is only indicated for the data collection used to predict user interests to deliver online advertising. These principles do not apply to websites that collect that information for its own uses. 2 years later, the Digital Advertising Alliance (DAA) announce an expansion of the program in order to

include the non-advertising businesses to the self-regulation [75]. These new principles prohibit third parties to collect, use or transfer any multi-site information. However, these data was mostly covered in the areas of insurance, credit, employment or health.

Despite the fact that many regulations exist, there is not a continuous control of the websites to check if they are actually following them. Creating a organization responsible for this would secure the compliance of regulations and therefore improve the privacy control of the users.

2.5 Discussion and Concluding Remarks

Since the introduction of cookies, web tracking has evolved along with the techniques to evade it. New advanced stateless device fingerprinting techniques have been proposed and adopted by trackers, including well-known advertisement and analytics companies, as recent work discovered [14, 13]. These advanced methods not only avoid the necessity of storing on the client's machine but also, since they are not contemplated in current regulations against web tracking, they are able to operate with no regulation even though its purpose is the same as cookies.

Due to the proliferation and the awareness that recent work have raised in the community, new techniques for detection of web tracking behavior have been proposed. Besides common blacklists, that can be subverted easily, heuristics [80] and rule-based systems [235, 215, 14, 13] have also been proposed by the community. These methods provide a way to detect these web tracking techniques and block them if necessary. These methods have been primarily utilized to perform studies about the current ecosystem of web tracking in the Internet, raising a general concern about the use of these methods. However, they can be adopted in the future in order to be used as complement of current blacklisting methods to ensure reactive protection of users against web tracking methods.

More proactive methods have also been proposed such as privacy-aware browsers, standardization, or regulation. In web browsers, even if randomizating the parameters that these techniques usually employ for generating the unique identifiers is a proposal, if detected, may also be used by web trackers as a parameter for generating an unique fingerprint. Therefore, other methods should be explored for a complete user protection against web tracking. Standardization and regulation policies are an important effort to fight against the generalization of web tracking and also the users'

lack of knowledge about being tracked and fingerprinted. Even though they are important steps against tracking — and online surveillance in general — it should be complemented because new and more advanced tracking techniques may be developed, bypassing the restrictions imposed by laws or standards.

2.6 Summary

Web tracking is a commonly-used practice on the Internet devoted to retrieve user information for activities such as personalization or advertisement. These techniques are said to drive the web economy, although they are commonly used to invade users' privacy. In the last years, a general concern raised about web tracking, looking forward to combat it in many ways like regulations, anti-tracking methods and even standardization. In this chapter, we analyze and discuss the current techniques for web-tracking as well as techniques for its detection and analysis, and countermeasures to prevent web tracking.

Part II

Web Security and Privacy Analyses

"A man is a success if he gets up in the morning and goes to bed at night, and in between does what he wants to do."

Bob Dylan (1941–)

Chapter 3

Knockin' on Trackers' Door: Large-Scale Automatic Analysis of Web Tracking

Given the background presented in the previous part, in this chapter, we present the first large-scale analysis of generic web tracking scripts. Due to the limitations of currently available solutions, we build our own tracking analysis tool called TrackingInspector. In contrast to existing solutions, based either on blacklists or static rules, this tool is based on code similarity and machine learning. TrackingInspector automatically detects known tracking script variations and also identifies likely unknown tracking script candidates. We use TrackingInspector to analyze the Alexa top 1M sites [252] to answer the following research questions: (i) how widespread is web tracking on the Internet?, (ii) what is the current ecosystem in web tracking provision and deployment on the Internet?, (iii) can current blacklisting solutions limit or block most web tracking on the Internet?, and (iv) can TrackingInspector discover new web tracking scripts? To what extent?

The major contributions and findings of this chapter are three. First, we performed the first large-scale study of generic web tracking. More than

90% of the websites performed tracking and more than 50% of the scripts exhibited a tracking behavior. Second, we present TrackingInspector, the first tool to automatically detect generic tracking scripts through code similarity and machine learning. The results show its ability to automatically detect known tracking scripts and their modifications, and to discover potentially unknown tracking. Our method was able to detect more than 2M known tracking script versions whereas current blacklisting solutions were only able to detect 64.65% of them in the best scenario. This indicates that many variations of tracking scripts are bypassing current solutions. Therefore, we studied these hiding techniques, finding several *script renaming techniques*. Finally, more than 5.5M not previously reported scripts exhibited a tracking behavior. These scripts include over 700 new unique potential device fingerprinting scripts: more than 400 performing canvas fingerprinting, more than 200 performing font probing, and more than 50 exhibiting both. TrackingInspector also automatically detected a previously reported targeted *fingerprinting-driven malware* campaign exhibiting fingerprinting behavior, present in two websites not reported as infected.

3.1 Tracking Analysis & Detection

3.1.1 General Description

It is important to remark that the definition of web tracking has been a controversial debate due to the different existing tracking types. In this work, we are going to use Ghostery's[1] definition, which is also used by many other security and privacy companies: *"Trackers (also commonly referred to as "tags") are snippets of code that send and receive information about you to companies to track and analyze your behavior, deliver ads, connect social media, provide comment sections, and more."*

Current Solutions

We evaluated the following solutions for tracking detection: blacklisting (e.g., *EasyPrivacy* [15] and *Ghostery* [57]), the EFF's tool *Privacy Badger* [101] based on simple heuristics, *FPDetective* [14], and *OpenWPM* [82].

Blacklisting tools rely on blacklisted script names, URLs, and domains. Although these methods detect the known tracking scripts and domains,

[1] https://www.ghostery.com/submit-a-tracker/

they fail to detect simple variations such as renaming the script or modifying the domain where the scripts are hosted. Besides, they may incorrectly block scripts devoid to tracking but named with the same script name as one within the blacklist.

Heuristics used in the *Privacy Badger* plugin block third-party cookies. This approach raises false positives and only focuses on cookies. Tracking adopts many different forms not covered by this tool.

FPDetective and *OpenWPM* are tracking analysis frameworks based on blacklisting and/or rules to detect tracking. These frameworks solve the main limitations of blacklisting. However, as these techniques are based on predefined rules, the tracking script can be modified with methods that bypass the defined criteria (see §9.6 for some examples).

TrackingInspector Solution

Our solution is composed of three components:

- A *Crawler* (§3.1.2) to retrieve web scripts and content.
- A *Script Database* (§3.1.3) with known tracking scripts.
- A *Text-based Analyzer* (§3.1.4) to analyze the scripts using the *Script Database*. It detects both known (or versions) and potentially unknown tracking.

TrackingInspector starts by downloading scripts through the *Crawler*. The *Crawler* waits until every script is downloaded; including third-party, first-party, or HTML-embedded scripts. Then, *Text-based Analyzer* analyzes them using its two components:

1 *Known Tracking Analysis* measures the similarity of each script with the ones stored in the *Script Database*. The script can be a version of a known tracking script or not. This analyzer is intended to be an end-user solution.

2 When no match is found, *Unknown Tracking Analysis* inspects the script using a machine-learning algorithm trained using the *Script Database* to detect likely unknown tracking scripts. This component is devoted to find new tracking scripts to be added to the database.

Dynamic approaches monitor the web behavior during browsing and compare it with specific rule sets for concrete types of tracking [14]. Instead, TrackingInspector uses the *Crawler* to dynamically retrieve all the scripts of the site and then, the scripts are analyzed by the *Text-based Analyzer*.

Script Representation

We tested two different approaches: Abstract Syntax Trees (ASTs) and Bag of Words (BOW). While ASTs represented the specific syntax of the functions within the code, the BOW approach captures the token frequencies to model the script. In our preliminary tests, the BOW text-categorization approach behaved better to detect generic tracking behaviors. AST approaches represent the code syntax strictly taking into account also the script structure, while BOW just represent the usage of tokens. Hence, ASTs are worse when dealing with script modifications or new scripts than the standard BOW model.

BOW models the scripts using a Vector Space Model (VSM), which represents them as vectors in a multidimensional space whose dimensions represent the possible tokens within the scripts. Scripts are represented as a vector of frequencies of tokens. Since our goal is not to capture the behavior but to detect any type of tracking, other approaches may overfit and fail to detect modifications or new web tracking scripts. BOW has been used for a broad range of web security problems such as detection of drive-by-download attacks [234], to measure the impact of different models in malware detection [47], or vulnerability detection [292].

3.1.2 Crawler

Overview

The *Crawler* component automatically retrieves every JavaScript file statically or dynamically loaded in a website. The *Crawler* is based on *PhantomJS* [132], a well-known headless browser. To avoid the detection of the automated browsing when using headless browsers, we adapted existing advanced hiding methods [253] to disguise the browser as a common one. With this adaptation, our *Crawler* is capable of performing transparent and exhaustive browsing. The *Crawler* also deals with obfuscation, and cleans the generated caches and cookies after each website inspection.

Methodology

The *Crawler* starts visiting the frontpage of the site under inspection. It saves all the downloaded scripts, taking into account if different scripts have the same name or if they are downloaded multiple times. To retrieve them, instead of waiting a fixed time and gathering them, like previous work on fingerprinting detection [14, 13], we analyzed the different script generation behaviors in 250,000 random websites within the Alexa top 1M websites to define a methodology for web analysis and set the adequate times to retrieve every script.

The resulting methodology starts with the *Crawler* waiting until all different frames in the website are loaded with a fixed maximum time of 60 seconds. If all frames are loaded before the 60 seconds are elapsed, the *Crawler* waits 15 seconds more (but never more than the maximum time). We added this second waiting time because in several cases additional scripts were downloaded after every frame had already been loaded because other scripts may have called them. If scripts are downloaded during this extra time window, the *Crawler* will wait until the remaining of the 60 seconds are elapsed. These time frames were selected taking into account the possible redirections within the website or its frames.

Then, the *Crawler* retrieves the resulting HTML code, with all the generated modifications, and gathers the HTML-embedded scripts. Then, the *Crawler* starts a deobfuscation phase and finally, cleans duplicate scripts, files that are not actually JavaScript, or empty files.

De-Obfuscation

We implemented a deobfuscator using *JSBeautifier* [173], a well-known deobfuscator as a starting point. Using the techniques implemented in this tool, our deobfuscator tries to unravel the original code, checking in each iteration whether or not the script has been deobfuscated with the specified metrics. Therefore, we can retrieve the original code conserving variable names and structure. Even if many approaches to obfuscate code exist, the analysis of Curtsinger et al. [63] found that the simple `eval` unfolding is the most commonly used, which is included in our deobfucator along with others.

In this way, we can deal with multiple layers and multiple known techniques of obfuscation. In fact, during our work, we found some obfuscators that use others iteratively to perform multiple layer obfuscation. When it is deobfuscated, our method performs an additional step to deal with one

obfuscator with a particular behavior. This particular one, when the free version is used, places the original code in an escaped string within the code while the rest of the code is used to call functions in the string and also performs a callback function to notify the authors that this script has been executed. Our method retrieves the code stored in the string and unescapes it, discarding its additional code.

Limitations

This approach may also have its shortcomings. In particular, a dedicated tracker can use more advanced obfuscation to bypass current *JSBeautifier*-based deobfuscation pass.

3.1.3 Script Database

It stores both known tracking and non-tracking scripts. In the case of the non-tracking scripts, we downloaded scripts from open-source projects and from randomly accessed websites (that did not belong to the Alexa top 1M sites), manually verified afterwards.

Data Sources

To generate the tracking scripts dataset, we retrieved scripts on the following sources:

- **Blacklists:** We used *EasyPrivacy*, *ABINE* [12], *Kaspersky Tracking List (ABBL)* [151] , the tracking version of *AdGuard* [16], the tracking list *FanBoy* [88], and the *Tracking Detection System (TDS)* by Rob van Eijk [275]. These lists were selected because they include scripts and not just domains. We omitted blacklisted domains because our goal is to detect tracking scripts rather than domains. However, this information was used in the large-scale analysis (along with other 6 tracking domain blacklists that will be detailed in §3.2) to compare the results and findings of TrackingInspector. Some of these lists included a whitelist composed of tracking scripts that are not considered harmful. Since our goal is to detect tracking behavior, we also included this type of scripts.
- **Open-source Tracking Projects:** We retrieved several open-source tracking projects: *BeaverBird* [250], *FingerPrintJS* [277], and the famous *Evercookie* [147].

- **Academic Papers:** We also included in our tracking *Script Database* the scripts found in [14, 13].

We processed the list of potential scripts and stored the scripts whose complete URL was available. When the scripts were not available, we tried to download them through `archive.org`, removing all the service-related text and code afterwards. When script names were only available, we searched it using several code searchers (e.g., *meanpath* [189], *Nerdy-Data* [65], *FileWatcher* [95]), or common search engines (e.g., *Google* and *Bing*). Then, we manually checked each script to determine whether they performed tracking or not.

Configuration

Since some of the scripts were obfuscated, we deobfuscated them as described in §3.1.2. Then, we removed duplicates or different versions of scripts. To this end, we modeled the code using the representation detailed in §3.1.1 and computed the cosine similarity of each script within the *Script Database*. To set a threshold for the detection of versions of original scripts, we conducted an empirical validation over the scripts and selected 85% because it was the lowest one (more coverage), raising no false positives. Then, we added the original scripts to the *Script Database* and also a small number of script versions that presented new functionalities to the original. After this process, 957 original tracking scripts were stored in our *Script Database*. We randomly selected 957 non-tracking scripts from the aforementioned sources.

3.1.4 Text-based Analyzer

Overview

The *Text-based Analyzer* is responsible for the detection of tracking scripts. This component is divided in two different sub-components: a (i) *Known Tracking Analysis*, responsible for the detection of versions, or modifications of known scripts stored in the *Script Database* and a (ii) *Unknown Tracking Analysis*, whose goal is to automatically identify tracking script candidates.

Text-based Web Tracking Detection

Although the goals of known and unknown approaches are different, they both represent scripts using the BOW approach. As in the *Script Database*,

the scripts are modeled through a VSM, composed of the terms within the scripts. *Text-based Analyzer* represents each script as a sequence of each term frequencies, using the well-known *Term Frequency [178] – Inverse Document Frequency* (TF–IDF) [259] weighting schema.

Known Tracking Analysis detects versions or modifications of currently known tracking stored in the *Script Database*. This component computes the cosine similarity of the script under inspection with known tracking. The cosine of the angle formed by the vector representation of the two scripts is the similarity — if they are totally equal the angle will be zero and their similarity 1, while when the angles are orthogonal and hence they do not share any token, their similarity score will be 0. When the empirically computed threshold of 85% similarity is surpassed, the script is flagged as a known tracking script version.

Unknown Tracking Analysis is based on supervised machine learning. The features used for the machine algorithm are the tokens in the BOW model previously explained. Machine learning develops algorithms to automatically learn behaviors from data. Since, in our particular case, data is labeled (i.e., tracking and non-tracking), supervised machine learning algorithms were selected for this task. These algorithms have been extensively used in web security tasks such as detection of malicious websites [46] or malicious JavaScript code [62]. We use the *Script Database* to perform a 10-fold stratified cross-validation experimental evaluation with several well-known machine-learning methods (e.g., Naive Bayes, Bayesian Networks, C4.5, SVMs, and Random Forest) to decide which classifier to use. The best classifier was *Random Forest* [133] configured with 950 *Random Trees*. This ensemble method for classification creates several decisions trees at training and chooses the classification based on the mode or mean of their partial classifications. In the training phase, the bagging technique is used to create the weak random tree learners. To build the aggregate, a similar but more general method is used to select the different high-level splits, also known as feature bagging.

Evaluation

We evaluated the performance of these two components of the *Text-based Analyzer* in terms of tracking script detection, prior to their usage in our large-scale analysis:

- **Known Tracking Analyzer:** Using 85% similarity threshold, did not report any false positives in our tests, being able to detect any ver-

sion of known tracking at least 85% similar to samples in the *Script Database*. Therefore, we believe that this component should be used in an end-user environment as a replacement of blacklisting techniques. We will compare it with blacklisting solutions in §3.2.

- **Unknown Tracking Analyzer:** We consider scripts not detected by the *Known Tracking Analyzer* as *previously unknown*. During the cross-validation evaluation, this component achieved an area under the ROC curve of 0.982, a true positive rate of 94.4%, and a F-measure (the harmonic mean of precision and recall) of 0.935. This method is the first one in the literature able to detect previously unseen tracking and devoted to the discovery of new tracking rather than using it in an end-user environment. This discovery of potentially tracking acts as a filter for a manual inspection to include actual tracking in the *Script Database*. To further evaluate the *Unknown Tracking Analysis* component, an additional evaluation of the method will be performed through a manual inspection of tracking scripts in the wild in §3.2.4.

To sum up, both components comply with our initial requirements to be used in the large-scale measurement and analysis.

3.2 Large-Scale Analysis

3.2.1 Preliminaries

The goal of this analysis is to answer the next research questions: (i) *how widespread is web tracking on the Internet?*, (ii) *what is the current ecosystem in web tracking provision and deployment on the Internet?*, (iii) *can current blacklisting solutions limit or block most web tracking on the Internet?* and (iv) *Can* TrackingInspector *discover new web tracking scripts and to what extent?*

The *Crawler* retrieved the scripts within the Alexa top 1M and the *Text-based Analyzer* inspected them. When downloading scripts from a removed site, we searched it through `archive.org`. Since `archive.org` adds code and also files to the website, we made a cleaning process. 3.67% of the websites were not accessed because its access was restricted (401 and 403), not accessible, or were impossible to find in `archive.org`.

Some of the scripts flagged as likely unknown tracking scripts by the *Unknown Tracking Analyzer* may be only unknown by TrackingInspector but known by existing domain blacklisting tools. To discriminate between

these two cases, we used the blacklisting tracking detection tools *EasyPrivacy*, *Kaspersky Tracking List (ABBL)*, *ABINE*, the tracking version of *AdGuard*, the tracking list of *FanBoy*, the *Tracking Detection System (TDS)*, *Ghostery*, *Privacy Badger*, *Disconnect* [76], *Truste* [273], *Privacy Choice* [268], and *Web of Trust* [216] (the domains classified there as the category 301 - online tracking).

We gathered data about the hosted website and the top-level domains where the scripts were downloaded. We also analyzed the domains, because the hosted scripts in domains influence the trust of the sites [213]. To correctly retrieve the top-level domain names, we used the list offered by Mozilla [102] (`effective_tld_names.dat`). Next, we extracted the country, the ISP, the associated ASs, and the web category of the domains. To determine the category, we used three services: *Cloudacl* [58], *Blocksi* [36], and *Fortiguard* [100]. Their category names are similar, and, after a normalization process, 78 category names were fixed. We also analyzed the reputation of the websites in the *webutation* service [282], that uses users' feedback, comments, and also different analyses such as *Google Safe Browsing*, *Norton Antivirus* or `phistank.com`.

For the sake of clarity, we define the terminology that we will be using:

- **Known Tracking Scripts:** We call known tracking scripts to those in the range between 85% and 100% similarity to the ones in the database.

- **Unknown Tracking Scripts:** Scripts that are less than 85% similar to the ones in the script database. These scripts can be either new scripts from scratch or versions of known ones modified enough to be considered *new*. Examples of unknown tracking scripts may include new implementations, added functionalities, or simply scripts never reported and thus never blacklisted. We will also refer to them as *tracking script candidates* or *likely tracking scripts* as the unknown text-analyzer has a small error percentage. We also distinguish between two sub-categories.

  - *Unknown Blacklisted Tracking Scripts:* Unknown scripts whose hosting domain is blacklisted but the script is not.

  - *Completely Unknown Tracking Scripts:* Unknown scripts whose hosting domains have not been blacklisted.

Table 3.1: Tracking and non tracking behavior prevalence in scripts.

Type	# Scripts	% Scripts
Tracking	11,984,469	57.15%
Non tracking	8,985,457	42.85%
TOTAL	*20,969,926*	*100.00%*

Table 3.2: Tracking and non tracking behavior prevalence in websites. % W. S. stands for the percentage of websites, considering only websites with scripts. % W. represents the percentage considering every website.

Type	% W. S.	% W.	# Websites
Tracking	97.58%	92.89%	894,779
Non tracking	2.42%	2.31%	22,220
No scripts	N/A	4.80%	46,277
Number of websites with scripts			*916,999*
Total number of websites			*963,276*

We also classify them as: (i) *Original*, if it is in the script database; (ii) *Unique*, if different compared by hash; and (iii) *Version/Sample* each download of a script.

3.2.2 Tracking Ecosystem

General Overview

20,969,926 script samples (tracking and non-tracking) were downloaded from the the Alexa top 1M websites. Impressively, nearly 60% of them were flagged as tracking (see Table 3.1). In other words, more than half the script functionality in the web is potentially devoted to track users. There were 46,277 websites with no scripts at all. Hence, we measure the tracking prevalence percentage both in every website and in websites with scripts (see Table 3.2), finding that nearly every website performed tracking.

Only a 20% of the tracking samples were known, whereas the unknown samples were the stunning majority. However, using the previously omitted domain blacklists, 41.11% of these scripts were in blacklisted domains (see Table 3.3), being unknown blacklisted scripts. The number of samples where neither the script nor the domain were blacklisted, was an impressive 46.90%. Tracking scripts were downloaded from 891,873 different domains.

Table 3.3: Detected tracking script distribution. *Domains* refer to top-level domains where the scripts are downloaded. *Unknown (blacklisted)* refers to likely tracking script candidates unknown in the Script Database but whose domain is blacklisted.

Type	# Scripts	# Domains
Known	2,439,835	540,369
Unknown (blacklisted)	3,923,615	7,455
Unknown	5,621,019	841,425
TOTAL tracking	*11,984,469*	*891,873*

Website Demographics

For a better understanding, we analyzed different aspects and tracking prevalence. To find the relevance of tracking in each analyzed feature (website, category/*webutation*, and country/network entity origin in domains), we ran several preliminary tests computing the differences of tracking ratios per website to find which ratio could discriminate between the features. The results showed that computing the number of websites with only tracking scripts per each studied feature eased the discrimination, while the number of websites with some tracking and the number of scripts per featured showed the overall behavior.

We computed the following ratios: (i) % of tracking scripts per script in the category, (ii) % of websites performing any type of tracking per website in the category, and (iii) % percentage of websites with only tracking scripts per website in each category. The website categories with the highest tracking percentage were *personal websites*, *hacking*, *spyware and adware*, *social networks*, or *peer to peer*. The top categories with only tracking were *malicious*, *questionable*, *unknown*, and *websites with adult content*. Despite it cannot be used to discriminate the maliciousness or greyness, *malicious* or *grey* sites tend to only include tracking.

The relation between websites with some or only tracking scripts with *webutation* hinted that the presence of only tracking affects the reputation of a website. 15.41% of the websites in the *Red* category and 15.31% in the *Yellow* category only used tracking, while only 6.45% and 5.95% of the *Grey* and *Green* categories did. Since users are not usually aware of web tracking, we believe that these results indicate that sites perceived as *bad* by users have a higher ratio of tracking than non-tracking, similar to what happened with categories.

Table 3.4: Domain distribution with regards to tracking. *Only Tracking* represents the top-level domains that only contain tracking scripts, whereas *Only non tracking* represents top-level domains with only non-tracking scripts. *Tracking & non tracking* represent the top-level domains that contain both tracking and non-tracking scripts.

Type	# Domains
Only tracking	98,359
Only non tracking	41,640
Tracking & non tracking	793,515
TOTAL	*933,514*

Domain Demographics

We also measured domains hosting tracking, non-tracking, and both tracking and non-tracking scripts (see Table 3.4). We analyzed domains to understand the provision of web tracking. In fact, previous work found the correlation between the scripts in domains and their nature [213].

We discovered that, similarly to what happened to websites with tracking scripts, domains usually also host both web tracking and non-tracking scripts. However, the percentage of them hosting solely tracking scripts is not negligible (10.54%). Regarding the relation of countries with their domains, several small countries surprisingly hosted only tracking. Likewise, we found cases of either AS owners, ASNs, or ISPs whose domains were only used to host tracking. Nevertheless, given the small number of domains, we consider them irrelevant.

As we did with sites, we studied the correlation between the *webutation* of the domain and its hosting of tracking scripts. We found that, as happened with websites, the presence of only tracking in domains affected the reputation: *Yellow* and *Red* represented 22.87% and 23.71% of the domains hosting only tracking scripts while *Grey* and *Green* categories only contained 11.23% and 9.44%.

3.2.3 Analysis of Known Tracking

To compare the detection capabilities of TrackingInspector with current tracking blockers, we measured the number of known script samples that blacklisting solutions would have blocked. From all the methods, we chose script name and domain blacklisting as the baseline because they provided a broader detection compared to the other alternatives. Another possible

Table 3.5: Unknown tracking downloaded from blacklisted domains prevalence in websites.

# websites	646,428
– % in websites with tracking	72.24%
– % in websites (considering sites with scripts)	70.49%

solution not used in tracking detection but used in other domains was also compared: code hashing.

Results show that script and domain blacklisting only captured 43.80% and 33.84% of the known tracking script versions, respectively. Combined blacklisting solutions blocked the 64.65% of the known tracking scripts while code hashing only would had captured 2.04% of the samples. These results show that current anti-tracking solutions are clearly not enough, not only to fight against completely unknown tracking scripts, but also against modified known tracking scripts.

Moreover, we measured the prevalence of tracking script versions in the *Script Database*. *Google* related scripts were the most popular: 60.90% of the samples correspond to their scripts, including 29.27% of samples with analytics capabilities. Among other scripts we can find: 20.92% regarding advertisement (*33Across*, *Pzyche*, and *QuantCast*), 3.49% from analytics (*Yandex Metrica* and *comScore*), and 2.00% social analytics samples (*FlCounter* and *Pinterest*). These ones were used by the 84.02% of the websites with scripts.

540,369 top-level domains hosted known tracking samples (the most popular one was `google-analytics.com`). Their scripts were present in 63.14% of the websites with scripts. The rest of the domains belonged to Google or to well-known advertisement services. The hosting of known tracking scripts follows a long-tail distribution with a small number of domains (or companies) representing the majority of script downloads.

3.2.4 Analysis of Unknown Tracking

Unknown Tracking Analysis In-the-wild Evaluation

TrackingInspector flagged more than 9.5M scripts as unknown tracking and more than 5M were not previously known by any blacklist. Albeit we already performed a 10-fold cross validation, we also performed an in-the-wild manual validation.

Table 3.6: 10 most popular blacklisted domains hosting unknown tracking scripts.

Domain	# Websites
facebook.com	177,443
akamaihd.net	176,616
googlesyndication.com	166,469
google.com	156,214
twitter.com	120,843
gstatic.com	114,153
facebook.net	86,299
googleusercontent.com	86,023
googleadservices.com	83,676
ytimg.com	72,571

Table 3.7: Distribution of unknown tracking candidates in domain types.

Domain	# Samples	# Uniques
HTML	4,145,542	2,744,244
1st Party	679,319	283,337
3rd Party	796,158	241,578
TOTAL	*5,621,019*	*3,245,238*

To this end, we extracted a statistically significant random sample from all the scripts flagged as unknown tracking. The sample was composed of 273 scripts, representing a 90% confidence level and a ±5% confidence interval. According to statistical sampling [97], confidence level measures the probability of the extracted sample to represent the entire dataset given a fixed confidence interval.

With a considerable manual effort, a tracking expert performed an exhaustive analysis of each sample both statically and dynamically, corroborating that 257 out of 273 scripts were actual web tracking (94.1%). These results are nearly the same as the ones obtained in the 10-fold cross validation described in §3.1 (94.4%).

Unknown Tracking in Blacklisted Domains

Unknown tracking scripts downloaded from blacklisted domains appeared in 70.49% of the websites with scripts (see Table 3.5). Only 7,455 blacklisted domains (see Table 3.6 for the 10 most prevalent) hosted 3,923,615 of this type. This number is much smaller than in the case of known or

Table 3.8: Popular clusters per domain type.

Domain	Cluster	% Scripts
HTML	Downloader	24.53%
	Statistics	13.47%
	Social Sharing	3.33%
1st Party	Statistics	26.70%
	Stateless Tracking	15.98%
	Advertisement	11.16%
3rd Party	Statistics	20.86%
	Stateless Tracking	15.27%
	Advertisement	14.08%

completely unknown tracking scripts. In particular, script samples from `facebook.com` were present in 18.42% of the websites with scripts.

New Unknown Potential Tracking

Scripts & Domains Unknown tracking candidates represented 58.89% of the likely unknown tracking samples flagged by the *Unknown Tracking Analysis*. Their presence varied with regards to whether the domain was blacklisted or not. The prevalence of completely unknown tracking was higher than blacklisted ones: 90.69% of the websites with scripts used unknown tracking (see Table 3.9).

Due to the high number of discovered scripts, we performed an analysis to understand their nature. To this end, we measured the number of unique scripts and the domain type (third-party, first-party, and HTML-embedded) and performed a clustering analysis to find the most common behaviors.

We removed identical versions through code hashing and measured the number of unique scripts in each domain type (see Table 3.7). Usually tracking is used with third-party domains, but we found that is not the case for completely unknown tracking. Most of them (73.75%) were embedded in the HTML, bypassing current blacklisting solutions. 57.73% of the completely unknown were unique by hash. Only a small number of them repeated versions across different domain types (1st and 3rd party, usually), indicating that is not a common practice.

We computed the prevalence of domains hosting unknown tracking. The 10 most popular domains (for the list see Table 3.10) were different e.g., CDNs, search engines, social networks, or advertisement companies.

Table 3.9: Completely unknown tracking prevalence in websites.

# websites	831,677
– % in websites with tracking	92.95%
– % in websites (considering sites with scripts)	90.69%

Table 3.10: 10 most popular top-level domains hosting previously unknown tracking script candidates.

Domain	**# Websites**
disquscdn.com	15,185
vk.me	9,228
baidustatic.com	4,848
kxcdn.com	4,189
adformdsp.net	2,958
jivosite.com	2,829
yandex.net	2,739
st-hatena.com	2,399
gtimg.cn	2,384
bitrix.info	2,374

Clustering To understand the 3,245,238 unique tracking candidates, we performed a cluster analysis. To overcome the high overhead of performing a cluster analysis directly on the large number of scripts, we conducted a clustering analysis in two steps. In the first step we clustered the 957 known trackers in our *Script Database* to find the different categories. We chose the *Affinity Propagation* clustering [110] due to its ability to automatically compute the cluster membership of an unknown sample and because it does not need to specify the number of clusters.

In the second step, we computed the closest cluster for each of the different unknown tracking script candidates through the previously calculated clusters. The most popular category contained *downloader* scripts which were the 16.53% of the tracking candidates. The second most popular cluster included 12.85% of the scripts and was formed of *statistics* tracking scripts. We also measured the clusters with regards to their hosting (see Table 3.8).

3.3 Case Studies

Script Renaming Techniques

Blacklisting techniques only blocked 43.80% of the known scripts versions TrackingInspector detected due to *script renaming*. Over 35% of the samples of each original scripts changed their name. While samples of 200 of the original scripts did not present any change, versions from other 95 original scripts always changed their name. We analyzed three scripts: `piwik.js`, `evercookie.js`, and `dota.js`. 58.33% of `evercookie.js` samples, 91.13% `piwik.js` versions, and 99.66% of `dota.js` changed their name.

Among the script renaming techniques, we defined the following categories: (i) *related script renaming*, (ii) *random/neutral script renaming*, (iii) *functionality script renaming*, and (iv) *misleading script renaming*. *Related script renaming* changes the name to one directly or indirectly related to another service or website using the original script. For example, some versions changed their name to `chrysler.js` and `dodge.js`. *Random/neutral script renaming* replaces the name randomly, such as `penguin2.js` and `welcome.js`. *Functionality script renaming* modifies the name describing their goal, e.g., `fingerprint.js`, and `tracking.js`. Finally, *misleading script renaming* scripts change their names to well-known script names e.g., `jquery.alt.min.js` and `j.min.js`.

Canvas and Font Fingerprinting

Canvas fingerprinting [199] and font probing [79] are two device fingerprinting techniques. Due to their relevance, we studied them to determine how many unknown scripts of this type our tool found.

We implemented two experiments for each type. These were designed to filter the potential tracking script samples that match rules for font probing and for canvas fingerprinting. We built two set of rules based on the scripts found by [14] and [13]. 710 unknown unique device fingerprinting scripts were found: 408 show canvas fingerprinting behavior, 247 font probing behavior, and 55 both.

We also used the new device fingerprinting scripts as the *Script Database* and performed a *Known Tracking Analysis* to measure their prevalence in the Alexa top 1M as well as the domains used for hosting them (see Table 3.11). 28,632 samples were detected and 27,818 websites used these unknown scripts.

We also analyzed the top prevalent potential device fingerprinting un-

Table 3.11: Potential unknown device fingerprinting prevalence.

Type	# Scripts	# Websites	# Domains
Font	25,502	24,873	704
Canvas	2,810	2,776	290
Shared	320	320	45
Total	*28,632*	*27,818*	*1,037*

known script. The most prevalent unknown font probing script was `buttons.js`. It was hosted by *sharethis.com*, a well-known social widget for sharing content in social networks. The most used canvas fingerprinting unknown script was `Admeta.js`, downloaded by 981 websites from the domain*atemda.com*, an ad provider. Regarding scripts containing both techniques, `image.js` was the most common unknown script, downloaded by 131 websites from the domain `magnuum.com`, a content delivery network.

Fingerprinting-driven Malware

Some reports [249, 271] linked targeted malware campaigns with fingerprinting. According to our results, sites with only tracking tend to be questionable and malicious. Hence, we inspected domains hosting only tracking and discovered suspicious scripts. For example, a script performed a fingerprinting step that included identification of the browser, *Java*, *Flash Player*, *SilverLight*, and the presence of a Chinese antivirus and then performed suspicious calls. It is important to remark, that the malicious script discovery was performed manually, based on the fingerprinting behavior and our findings, but we did not build any specific detection technique.

`101.99.68.18` and `202.172.54` hosted the script. `101.99.68.18` was allocated in the ISP *Piradious-NET* and `202.172.54` in *M1 Connect Pte. Ltd.*, known for hosting malware. The script was `jquery.min.js`, as in the library *jQuery* and was not obfuscated. Two Chinese sites in the Alexa top 1M (`52lif-e.cc` and `examres.com`) contained the iframe used to gather the malicious script.

By searching the iframe name, we found that this script was part of the *Chinad* botnet, as reported by *MalwareBytes* [180, 181]. We did not perform a manual malware analysis, since it has already been performed by them. This is a exploit kit that compromised Chinese sites to fingerprint users looking for vulnerable components: e.g., in *Java* (CVE-2011-3544 and CVE-2012-4681), *Internet Explorer* (CVE-2014-6332), and *Flash*

Table 3.12: Comparison with Related Work since 2012.

Approach	Type	Stateful	Stateless	Variations	Unknown	Size
Blacklisting Solutions	Generic	✓	✓	✗	✗	–
Roesner et al., 2012 [235]	Specific	✓	✓	✗		2,098
Cookieless Monster [215]	Specific	✓	✓	✗		10,000
FPDetective [14]	Specific	✗	✓	✗	✓	1,000,000
The Web Never Forgets [13]	Specific	✗	✓	✗	✓	1,000,000
TrackingExcavator [167]	Specific	✓		✗	✓	10,000[a]
Englehard et al, 2016 [83]	Specific	✓		✗	✓	100,000/1,000,000[b]
TrackingInspector	Generic	✓	✓	✓	✓	1,000,000

[a] They analyzed 500 websites per year in the period 1996-2016.
[b] 100,000 stateful and 1,000,000 for stateless fingerprinting.

(CVE-2015-0311) and downloads versions of the *Chinad* botnet to perform DDoS attacks. To the best of our knowledge, the websites where we found the exploit kit were not previously reported.

3.4 Related Work

Previous work

Due to the concern that web tracking raised to users' privacy, a hectic research has been done to analyze these techniques.

One of the first tracking analyses that included HTML cookies was the one performed in [158]. Following this work, Mayer & Mitchell [186] studied different techniques for tracking including their policies and developed a tool to measure web privacy. Roesner et al. [235] presented a taxonomy for third-party tracking using cookies, measuring their presence. Nikiforakis et al. [215] studied three known fingerprinting companies and discovered that 40 websites within the Alexa top 10K sites used techniques such font probing. Acar et al. [14] discovered 404 sites in the top 1M using JavaScript-based fingerprinting and 145 sites within the Alexa top 10K sites using Flash-based fingerprinting. Also, Acar et al. [13] found a 5% prevalence of canvas fingerprint in the Alexa top 1M sites. They also found respawning by Flash cookies on 10 of the 200 most popular sites and

33 respawning more than 175 HTTP cookies. In the topic of web vulnerabilities, a previous analysis of the JavaScript included [213] determined the correlation between the trust of the included scripts as well as the domains hosting the scripts. However, our work differs from this large-scale study since we focus specifically in web tracking rather than vulnerabilities. Lerner et al. [167] presented *TrackingExcavator* and performed a retrospective analysis of tracking evolution since 1996, showing that it increased over time. Englehardt & Narayanan [83] analyzed the Alexa top 100K websites with regards to stateful tracking and the top 1M regarding stateless fingerprinting using blacklists and static rules, finding new device fingerprinting techniques. Our work differs in many ways (see Table 3.12 for a detailed comparison). We use the following parameters for the comparison:

- **Specific/Generic:** Specific approaches focus on concrete web tracking types using ad-hoc heuristics. A generic approach detects web tracking using a single approach for every type of tracking. Although some previous works deal with both stateful and stateless web tracking, we consider generic approaches the ones that do not rely on specific solutions or heuristics for each type.
- **Stateful/Stateless tracking:** We divide previous approaches into the ones devoted to detect classic stateful approaches, the ones that detect stateless ones, or the ones that detect both types.
- **Detection of Variants:** TrackingInspector detects variants of the known scripts. Other works, normally the ones based on heuristics, may also detect variants but they could not classify them as so.
- **Detection of Unknown:** TrackingInspector also detects potentially unknown tracking. Some heuristics-based methods are also able, but just when the script follows the defined heuristics.
- **Size:** To compare the soundness of our study, we measure the size in websites.

Summarizing, we believe that the major breakthroughs of our work are the following. First, TrackingInspector, a truly generic web tracking detector that does not depend on blacklists or specific rules but on the previously known tracking scripts. Second, we discovered more than 3 million new unique tracking script candidates, a number higher than any reported previous work. We also found 710 new potential device fingerprinting scripts: 408 canvas fingerprinting, 247 font probing, and also 55 that

```
1  //https://browserleaks.com/canvas
2  var ctx = canvas.getContext('2d');
3  var txt = "exampleText. <canvas> 1.0";
4  ctx.textBaseline = "top";
5  ctx.font = "14px 'Arial'";
6  ctx.textBaseline = "alphabetic";
7  ctx.fillStyle = "#f60";
8  ctx.fillRect(125,1,62,20);
9  ctx.fillStyle = "#069";
10 ctx.fillText(txt, 2, 15);
11 ctx.fillStyle = "rgba(102, 204, 0, 0.7)";
12 ctx.fillText(txt, 4, 17);
13
14 //Extracting image (detectable)
15 md5(ctx.getImageData(0, 0, canvas.width,
       canvas.height).data.toString())
16 md5(canvas.toDataURL())
17
18 //Extracting image (undetectable)
19 imageData = ""
20 for (i = 0; i < canvas.width/10; i++) {
21   for (j = 0; j < canvas.height/10; j++) {
22     imageData += ctx.getImageData(i*10, j*10, 10, 10).data.toString()
23   }
24 }
25 md5(imageData)
```

Figure 3.1: Snippet of OpenWPM detectable and undetectable canvas fingerprinting image extraction techniques.

exhibited both fingerprinting behaviors, a number also higher than previous work. Our tool also automatically detected the fingerprinting phase of a targeted malware campaign for the first time to date.

Empirical comparison

We also implemented a comparative experiment that consist of 2 specific case studies, showing how our tool can detect instances of tracking scripts that are currently not flagged by other tools.

Canvas Fingerprinting. Englehard et al. [83] analyzed Alexa top 1M websites, using OpenWPM [82], looking for tracking scripts (canvas finger-

printing included). For this specific technique, they proposed a set of 4 rules as a filtering criteria. Even if theses rules could definitely help in the detection process, there is an extremely simple evasion that any script could perform in order to completely avoid detection. The exact rule that allows it, is the following: *"The script extracts an image with `toDataURL` or with a single call to `getImageData` that specifies an area with a minimum size of 16px x 16px."* If the script performs multiple calls to `getImageData` without exceeding the size determined in the rule, the analysis will erroneously flag the script as non-tracking (see Figure 3.1). In order to check if our approach would detect this case, we prepared two different scenarios.

1 We modified a script known for performing canvas fingerprinting (`dota.js`) to obtain the data following the aforementioned evasion technique. Then, we performed a *Known Tracking Analysis* to see if this change would be enough to evade the detection of the script as a variation of the original script. TrackingInspector was able to identify the script with a similarity of 99% and flags it as a modification of a known tracking script present in the *Script Database*.

2 In this case, instead of using a known script as a baseline, we created a new tracking script from scratch, but based on the same idea. As in the previous scenario, we implemented the data retrieval process following the evasion technique described. We performed an *Unknown Tracking Analysis* in this script to verify the effectiveness of the tool in this exact case. TrackingInspector was able to correctly classify the script as an unknown tracking script.

In conclusion, in both of the presented scenarios, our tool was able to detect the tracking script using its different components, in one case to detect the variation and in the other to detect the fingerprinting technique; while the previous approaches failed.

Font Fingerprinting. This type of fingerprinting technique has been extensively analyzed using both OpenWPM [83] and FPdetective [14]. One of the criteria used in these works to classify the script as a candidate of performing this kind of tracking, is the number of different fonts loaded (50 and 30, respectively). However, the work of Fifield and Egelman [94] allows to generate a specific type of font fingerprinting that measures the bounding boxes of some specific Unicode code points. For this case, the technique just needs five generic families (e.g., sans-serif, serif, monospace, cursive,

and fantasy). As the number of different fonts used is much lower than the ones specified in their respective rules, neither of the previously mentioned analysis were able to detect them. In contrast to the previous case, this is not a small modification of the technique to avoid detection, but another font fingerprinting approach.

Therefore, we have created a fingerprinting script following this font fingerprinting technique from scratch. Then, we performed an *Unknown Tracking Analysis* to verify if our tool was able to detect this script as a tracking script. TrackingInspector correctly classified this script in the tracking group. As our approach is not based in strict human-generated rules, but in a learning progress with known scripts performing various types of tracking techniques, it can detect different fingerprinting attempts without explicit rules.

3.5 Conclusions

TrackingInspector measured the web tracking prevalence in websites and domains. The results show that web tracking is very extended and that current solutions cannot detect every known or unknown tracking. We also examined the hiding techniques used to avoid blacklists, finding different script renaming techniques. TrackingInspector detected both known or variations of tracking and likely unknown web tracking. We also found new potential stateless device fingerprinting and their prevalence, showing that even well-known companies provide them. Among the discovered unknown web tracking, we found a previously reported malware campaign targeting Chinese websites, showing that malicious activities may exhibit fingerprinting behavior. We believe that *fingerprinting-driven malware* may become a relevant issue in the future.

"If I'm going to be a caged bird, I'll sing the best song I can."

Wes Craven (1939–)

Chapter 4

The Onions Have Eyes: A Comprehensive Structure and Privacy Analysis of Tor Hidden Services

Informally, the *Dark Web* refers to the small portion of the *Deep Web* (the part of the Web which is normally considered to be beyond reach from current search engines) based on *darknets*. Common darknets include, among other smaller P2P networks, *FreeNet* [3], the *Invisible Internet Project* (I2P) [5], and *Tor* [7]. In the case of Tor, Tor hidden services are used to provide acccess to different applications such as chat, email, or websites, through the Tor network. In this chapter, we focus in particular on the analysis of *websites* hosted on Tor hidden services — due to Tor's much larger popularity between users, which comprised around 7,000 relays or proxies by the time of this writing [9]. The Tor network is based on the onion routing technique [233] for network traffic anonymization.

Due to its hidden nature, Tor hidden services are used for a large range of (cyber)-criminals activities [55, 56, 258, 239]. Thereby, several studies [33, 179, 128, 175] focused on how to discover, access, crawl, and categorize the content of the Dark Web.

Recently, *OnionScan* [168, 171, 170, 169] and the *DeepLight* reports [137] have analyzed some features related to the content, the size, and the connectivity of the Dark Web. While these studies have helped to better understand its nature, we still lack a complete analysis of Tor hidden services to compare their structure with the corresponding studies of the *Surface Web* [41, 191].

Similarly, while the research community has put a considerable effort to analyze the privacy and security of Tor relays [187, 50, 287, 240] and of its routing protocol [193, 142, 263, 160], a comprehensive analysis of the privacy implications at the application level and of the prevalence of fingerprinting and web tracking is still missing (although these subjects have been extensively studied for the Surface Web [215, 14, 13, 162, 167]).

To fill these gaps, in this chapter we present the most comprehensive structure and privacy analysis of the Tor hidden services. Our work is divided in three parts. In the first, we present the most complete exploration of the websites hosted on the Tor hidden services performed to date. Previous measurement studies were limited just to the home pages of each site. While it is true that 80% of the websites have less then 18 URLs, according to our experiments their home pages contain only 11% of the outgoing links, 30% of the resources, 21% of the scripts, and 16% of the tracking attempts. To overcome this limitation, in our analysis we exhaustively downloaded all the reachable content for over 80% of the websites (for a total of 1.5M pages), and we completely crawled 99.46% of the sites to extract links to other domains.

In the second part of this chapter, we present an analysis of the collected data looking at links and redirections, as well as at the external resources imported by onion domains from the Tor hidden services themselves and from the Surface Web. In addition, we perform a complete structure analysis of the three connectivity graphs – links, resources, and redirections – and compare them to previous structural analyses conducted for the Surface Web.

Our experiments show that Tor hidden services are highly connected and that their internal structure is sparse, with a high number of strongly connected domains. Overall, 10% of the websites have no incoming links and a stunning 98.8% of all the discoverable domains are already included in public directories (with a single one - `tt3j2x4k5ycaa5zt.onion` pointing to over 70% of the websites we visited).

Quite surprisingly, we also discovered that Tor hidden services are more

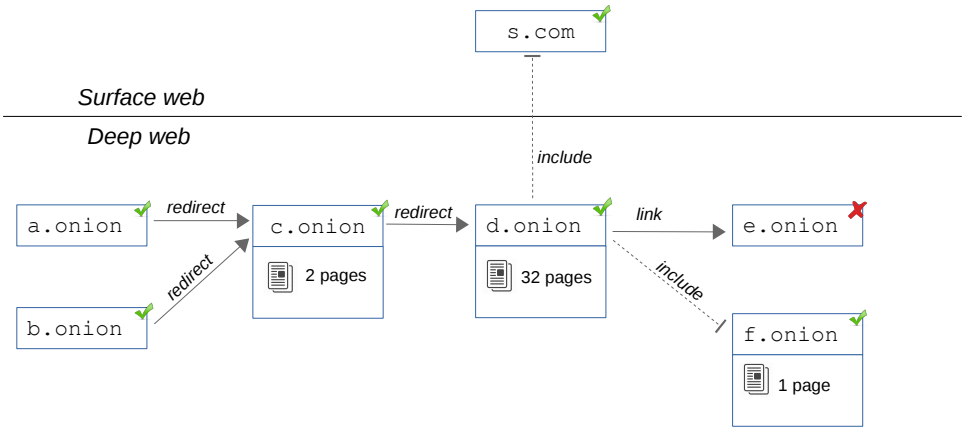


Figure 4.1: Tor Hidden Services Architecture Example and Clarification.

connected to the Surface Web than to other Tor hidden services. In particular, over 21% of the onion domains import resources (e.g., Javascript files) from the Surface Web. Due to these external components, we discovered that Google alone can monitor the accesses to almost 14% of the Tor hidden services in our dataset.

Since these connections can raise some privacy concerns, in the third part of the chapter we analyze the privacy implications of the structure of Tor hidden services and we measure for the first time the prevalence and nature of web tracking in this environment. Using a generic web tracking analyzer, we discovered that, despite the fact that the usage of scripts in Tor hidden services is smaller than in the Surface Web, the percentage of them used for web tracking is similar. More than 75% of the onion domains that contain at least a Javascript file, perform some form of tracking. Moreover, we have found that the majority of web tracking present in Tor hidden services is not known by any anti-tracking technique.

Another interesting finding is the fact that over 30% of the tracking and fingerprinting in Tor hidden services uses scripts imported from the Surface Web. This is particularly worrying, as it may be used to follow known users when they visit anonymous websites on Tor. Finally, we discuss how the owners of the websites try to hide their tracking attempts, for instance by performing tracking in the middle of a redirection chain.

The remainder of this chapter is organized as follows. Section 4.1 details the analysis platform and the methodology used in our Tor hidden

services analysis. Section 4.2 describes the conducted structural analysis of the onion domains, as well as our findings. Section 4.3 discusses the privacy implications of our previous findings, and details the specific web tracking analysis performed in the Tor hidden services. Section 9.6 provides the context of this chapter given the current previous work. Finally, Section 9.7 summarizes the main conclusions.

4.1 Analysis Platform

While the connection among different web pages is part of the nature of the Surface Web, web sites in the Dark Web are often more ephemeral and isolated among one another. This difference makes crawling the Dark Web a non-trivial task, that goes beyond simply navigating through hyper-links.

Therefore, to perform our study we manually collected a list of URLs associated to 195,748 onion domains from 25 public forums and directories. We then implemented a custom crawler to explore this seed list to collect data for our analysis and extract new domains. Our crawler can be configured to operate according to two different behaviors. In *"collection mode"* the crawler retrieves all the HTML and Javascript resources of the target domain. This mode has restrictions regarding the maximum depth and number of links it can explore for each onion domain. When these thresholds are reached, the system switches to *"connectivity mode"*, where it simply crawls the remaining pages looking for new links towards other onion domains. While in this mode, the system does not store a copy of the resources and therefore it is not restricted in its depth level (but with a maximum of 10K URLs per domain) and can visit a much larger number of pages of the target domain.

After the crawler has collected all the data, we performed an offline analysis – divided in two parts:

1 **Structure Analysis:** the goal of this analysis is to study the number of connections and their nature to better understand the overall structure and properties of the Dark Web. For instance, we measured the prevalence of links, resources, and redirections both towards other onion domains and towards the Surface Web. We also built the complete graph for each connection type and performed a complete structure analysis, comparing the results with those obtained by analyzing the properties of the Surface Web.

2 **Privacy Analysis:** The structure analysis raised several privacy concerns, which we analyze in more details in this second analysis phase – which focus on studying the tracking ecosystem in the Dark Web. Since our goal is to obtain a general overview of web tracking in the Dark Web, rather than focusing on a specific type of tracking, we did not implement our own tracking detector but we reused instead a generic tracking analyzer capable of detecting both known and unknown tracking techniques [242].

Finally, it is important to remark that during our analysis we distinguish between *domains* and *URLs*. Figure 4.1 clarifies the distinction. For instance, the figure shows seven domains (`a.onion`, `b.onion`, ..., `f.onion`, and `s.com`). Domains can be hosted either on the Surface Web, or on the Dark Web (onion domains). Some of them point to website hosting actual content (e.g., `c`, `d`, and `f`) while other only serve to redirect users to other domains (such as `a` and `b`). For our analysis purposes, we define the *size* of a domain as the number of unique URLs served by that domain that returned HTML content. Domains can also host other resources (such as JavaScript, pictures, and CSS file) which are valid URLs but do not count as accessed URLs nor for the computation of the size.

4.1.1 Design and Implementation

We implemented our Dark Web crawler on top of the headless browser *PhantomJS* [6]. To prevent websites from easily identifying our crawler, we implemented a number of advanced hiding techniques already used by other systems.[1] Moreover, we cleaned cookies and the browser cache after visiting each website.

The crawler receives the main URL of an onion domain and retrieves its HTML and external scripts. To accommodate for dynamic content, the system saves the content of each page after a certain time has passed. Since the Dark Web is considerably slower than the Surface Web, instead of using a fixed waiting time, we first performed an experiment to measure the appropriate value. For this test we extracted a random sample of 25% of the initial seed domains and analyzed the loading time of their pages. Based on the results of our experiment, we configured our crawler to wait for five additional seconds after a page is completely loaded, and for a maximum of 60 seconds otherwise.

[1] `https://github.com/ikarienator/phantomjs_hide_and_seek`

To deal with script obfuscation, we implemented a de-obfuscator using *JSBeautifier*.[1] that iteratively tries to deobfuscate a page, testing at each iteration whether or not additional code has been revealed. In this way, we can also deal with multiple-layer obfuscation.

As already mentioned above, our crawler can work in two modes:

- **Collection mode:** In this mode the crawler collects and stores a large amount of data for further offline analysis, including all the HTTP headers, the HTML code, all scripts (both imported or embedded in the HTML), the list of performed redirections along with their origin, destination and nature (e.g., HTTP or JavaScript), and all the links (either visible or invisible) within the website. To mimic the behavior of a real user, we modified the `referrer` at each step to point to the correct origin URL. While in collection mode, the crawler only recursively visits ten links for each onion URL. To prioritize the links that can lead to more "interesting" pages, we first test the destination URL against a list of over 30 keywords (e.g., login, register, account, or password). Second, we dynamically compute the appearance of the link in the website according to the CSSs and other styles defined in the HTML. We use this information to rank the links based on their computed visualization size, weighted by its closeness to top of the website. Finally, the crawler in collection mode is limited to visiting internal links up to a depth of 3 layers from the homepage. In other words, the crawler collects data from the homepage, from ten of its internally linked pages, then from other ten starting from each of them, up to a depth of three steps.

- **Connectivity mode:** The "connectivity mode" extracts all the links (either visible or invisible) in the target website through a breadth-first exploration, without considering fragment ('#' position links), or files such as images or PDF documents. This mode is not limited by the 10-pages nor by the 3-level depth restrictions. Its major goal is to complete the crawling of a domain after the collection mode has reached its threshold. However, for practical purposes and to avoid getting stuck in endless websites that contain an infinite number of URLs (the often called *calendar effect*), we limited this mode to visit 10,000 distinct URLs for each individual domain.

[1] `http://jsbeautifier.org`

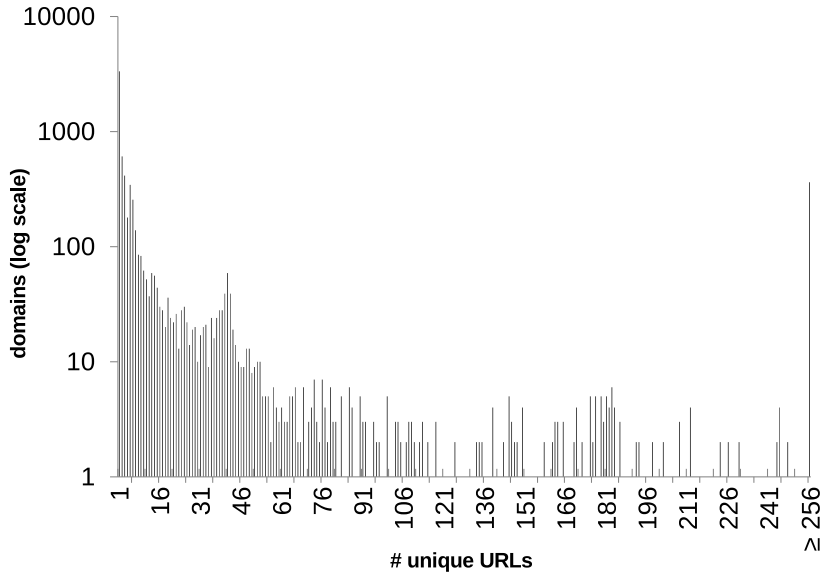


Figure 4.2: Distribution of URLs in each `.onion` domain.

4.1.2 Data Collection

Our collection started from an initial seed of 195,748 domains, retrieved from different sources: Tor gateways, `pastebin`, lists/directories (in the surface and dark web), `reddit` posts, and previous studies [137]. These type of sources are commonly used to discover hidden onion domains [56, 137].

Our crawler then visited each onion domain, initially running in collection mode. If this approach was unable to crawl the entire website, the crawler switches to the connectivity mode and visited the remaining internal links - this time just collecting other links without storing the content.

To gather the highest number of active domains possible, we performed our entire crawling experiment three times: twice in May 2016 (three days apart), and then again one month later in June 2016.

4.2 Structure Analysis

The first part of our study focuses on the analysis of the structure of the Tor hidden services. To this end, we analyzed the categories and languages of the retrieved websites, as well as their connections in terms of links, resources, and redirections. We also performed a graph analysis of the Tor

hidden services and compared the results with those of similar studies performed on the Surface Web.

4.2.1 Size & Coverage

From our initial seed of 195,748 domains, our crawler gathered a total of 198,050 unique domains. The small difference between the two numbers confirms the nature of the dark web, where websites are mainly reached from public directories or domains posted in forums (part of our initial seed). This has two important consequences. First, that existing directories already cover 98.8% of the discoverable domains. However, Tor hidden services also include websites intended for private use, that are never linked or publicized anywhere else – and that therefore cannot be covered in our study. Second, as we explain later in this section, our experiments show that 10% of the domains have no incoming links and therefore cannot be reached by a traditional crawler. While this percentage is very large, it means that the remaining 90% of the Tor hidden services are actually interconnected, and that therefore they are not just a collection of isolated web sites.

In our three consecutive crawling attempts, we discovered that only 7,257 were active domains. This is a consequence of the short lifespan of onion websites and of the fact that the majority of onion domains collected from public sources often become unreachable after a short amount of time. Therefore, public directories contain a very large amount of outdated material, which wrongly contributes to the image of Tor hidden services.

Interestingly, 81.07% of the active domains were completely crawled in "collection mode". In this case, our system was able to retrieve and store an entire copy of the website. An additional 18.49% of the onion domains were then crawled completely in "connectivity mode" (every link and connection to external domain was successfully collected) — and only for the remaining 0.54% of the websites our system was unable to visit all URLs because they contained more than 10K internal active URLs. Overall, our crawler accessed a total number of 1,502,865 unique onion URLs corresponding to 746,298 different base URLs (i.e., without taking their parameters into account). This is the largest number of URLs visited to date in a study of Tor.

A total of 203 onion domains only performed HTTP redirections to other domains without hosting any content. For the rest, Figure 4.2 shows a

Table 4.1: Most Popular Languages in Onion Domains.

Language	% Domains
English	73.28%
Russian	10.96%
German	2.33%
French	2.15%
Spanish	2.14%

log-scale distribution of the size of each domain. Quite surprisingly, almost half (46.07%) of them only contained a single page and over 80% contained less than 17 unique URLs. This means that vast majority of the websites in the dark web are extremely simple, and only few of them are large and complex applications, containing up to tens of thousands of different URLs.

4.2.2 Language & Categories

We additionally measured the distribution of languages and website categories within the active onion domains. Since the results are computed by analyzing each individual URL, each domain can be assigned to multiple languages and multiple categories (i.e., if a domain contains both English and Russian pages it is counted as belonging to both languages).

To obtain the actual language, we used the Google Translate API[1] and its language autodetection function. Overall, we found 63 different languages used in Tor hidden services. English was the most popular (present in 73.28% of the domains), followed by Russian, German, French, and Spanish (see Table 4.1). While the percentages are different, the ranking is very similar to the one of the surface web (English, Russian, German, Japanese, Spanish, French [264]) with the omission of Japanese and a larger percentage of English content. However, our results are different from the ones published in the DeepLight report [137] both in the number of languages and their distribution and ranking. This difference can be a consequence of our higher coverage, especially in the number of pages visited in each domain (for instance, a website may show only English content on the main page, but then provide also other languages by following specific links).

To identify the URL categories, we first used Google Translate to trans-

[1] https://cloud.google.com/translate/

Table 4.2: Categories in Onion Domains.

Category	% Domains
Directory/Wiki	63.49%
Default Hosting Message	10.35%
Market/Shopping	9.80%
Bitcoins/Trading	8.62%
Forum	4.72%
Online Betting	1.72%
Search Engine	1.30%

late the page content to English, we then removed stop words[1] and used a stemming algorithm to extract the root token of similar words (e.g, work, worker, and working).

Next, we modeled each URL as a *Bag of Words* [238] and performed a two-phase clustering. In the first phase, we randomly selected 10% of URLs to form the clusters by *Affinity Propagation* [110]. Then, in the second step, we computed the cluster membership of the remaining URLs. As a result, 79 different clusters were found. We manually inspected and assigned each of them to one of seven main categories (Table 4.2 shows their distribution). Overall, we found that 15.4% of the domains belong to more than one category. *Directory/Wiki* is the most popular category since many onion websites tend to include a resource repository of links, news, articles, videos or images. It is important to remark that directories do not necessary only link to external content, but they can also include their own. The second most popular category are websites containing a default hosting message such as "*This website is hosted by X*": this result was also observed by a previous measurement study [171].

4.2.3 Links, Resources, and Redirections

Table 4.3 shows the number, type, and prevalence of links present in onion domains, as well as the number of domains importing resources from other onion domains and from the Surface Web. Specifically, only 41.5% of the total onion domains contained HTML links to other onion domains and 40.6% contained links to the surface web. Regarding links to other onion domains, a stunning 68.34% of them were broken (i.e., the target was not

[1] Using the public list at `http://www.ranks.nl/stopwords`

Table 4.3: Links and Resources in Onion Domains.

Links	*to Onion*	# domains linking	3,013
		# domains linked	20,621
		# domains alive linking	2,482
		# domains alive linked	6,528
	to Surface	# domains linking	2,947
		# domains linked	83,984
Resources	*from Onion*	# domains importing	466
		# domains exporting	349
	from Surface	# domains importing	1,561
		# domains exporting	2,235

reachable during our experiments) confirming again the ephemeral and disconnected nature of websites hosted in the dark web. Only 6,528 from the 7,257 active domains in our dataset were found in the onion links, indicating that over 10% of the onion domains are isolated and unreachable from other domains in our dataset. Comparing the number of links to onion and surface domains, the number of surface links was clearly higher, even considering the inactive onion links.

Looking at the imported resources, only 6.47% of onion domains imported resources from other onion domains, while 21.51% of them imported resources from the Surface Web. Moreover, the absolute number of unique resources imported from the Surface Web is over five times higher than the resources imported from other onion domains. As we will discuss in more details in Section 4.3, this can have some privacy implications on users visiting the dark web using Tor Proxies.

Another relevant finding of our study is that only 36% of the onion domains we visited contained Javascript code (either embedded in the page or standalone). Interestingly, 48% of the URLs contained no scripts at all. This shows a different picture from the Surface Web, where current statistics report that almost 94% of websites use JavaScript [264].

Finally, we looked at redirections. Table 4.4 shows that, contrary to what we observed for links and resources, redirections performed by onion domains are more likely to target other onion domains: 3.90% of the onion domains contained at least one redirection to other onions, while 2.62% redirected to the Surface Web. This second group is particularly important because these redirections can be used to de-anonymize users in certain

Table 4.4: HTTP Redirection Distributions Performed in Onion Domains. Dest. means destination and D. domain.

Dest.	Type	# Source D.	# Dest. D.
Onion	HTTP	232	196
	HTML	37	22
	JS	17	12
	TOTAL	*283*	*225*
Surface	HTTP	117	124
	HTML	35	22
	JS	39	14
	TOTAL	*190*	*154*

configurations (e.g., those using a Tor proxy). Table 4.4 also distinguishes between HTTP and non-HTTP redirections. The majority of the sites used the `HTTP 30X` redirection method — 82% to redirect to other onion domains, and 62% to redirect to the Surface Web.

4.2.4 Graph Analysis

A study of the structure of the web was first conducted in 2000 by Broder et al. [41], and then revisited by Meusel et al. [191] in 2014. A first study about the connectivity of Tor hidden services has been recently presented [170], but the structure was not analyzed and only the main onion domains were taken into account by the authors. Hence, to provide a better comparison with the studies of the Surface Web, we performed a graph network analysis of the three types of connections between onion domains: links, resources, and redirections (Figure 4.3 shows the links graph, Figures 4.4a and 4.4b the in/out degree of the links, Figure 4.5a the resources graph, and Figure 4.5b shows the redirections graph), measuring the structural properties of the three resultant graphs.

The links connectivity graph contains 6,823 different nodes and more than 60,000 connections. The average shortest path to traverse the entire directed network was 3.697 edges and the largest (the diameter) was 49. This relation implies that it is highly connected but disperse, as we can also deduce from the high average connection degree and its low density (near zero). The graph has a high connectivity, containing 88.03% of strongly

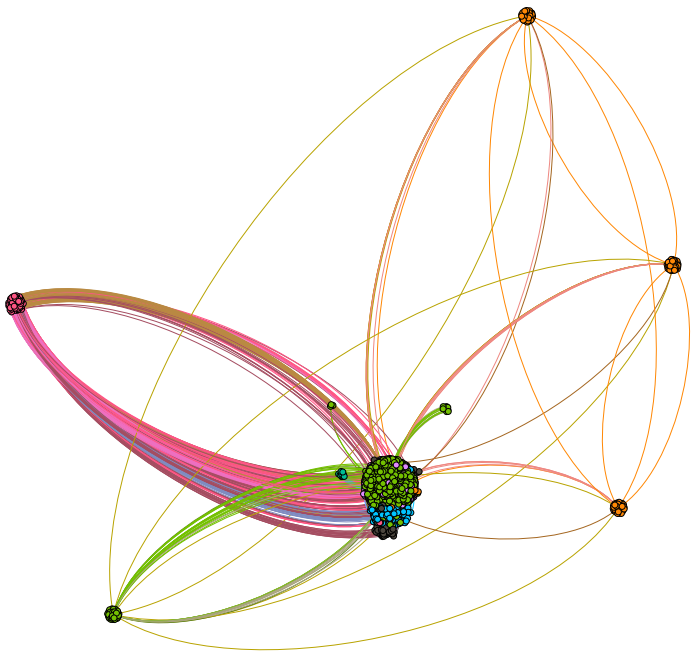
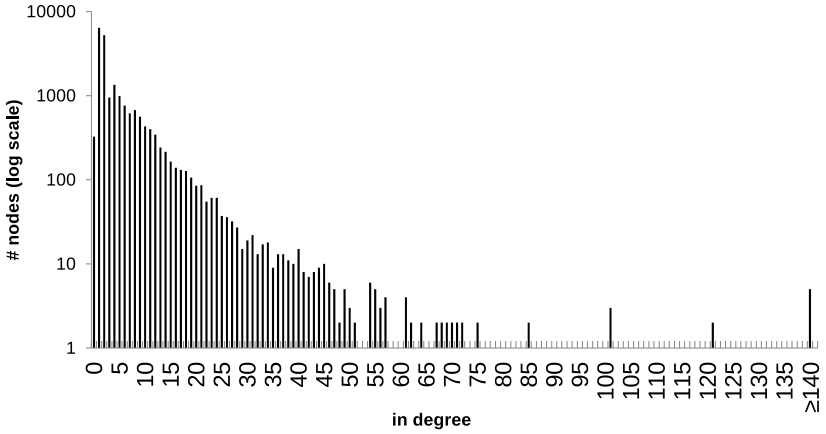


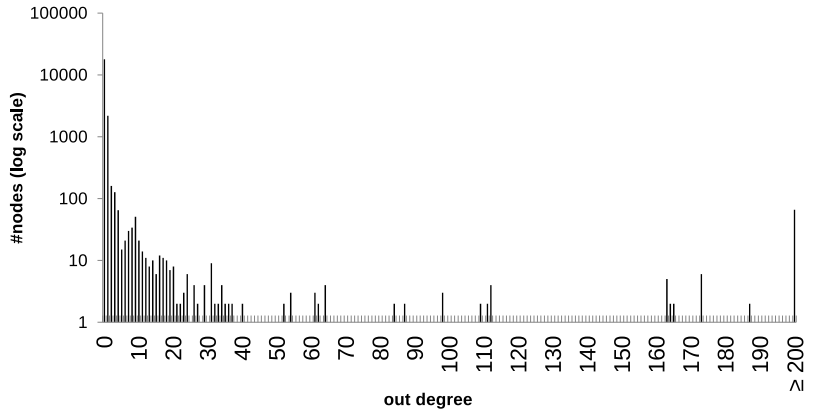
Figure 4.3: Links Graph of Onion Domains computed with the OpenOrd force-directed layout algorithm and colored communities through modularity. Isolated domains were removed from the figure for clearness of the representation.

connected nodes. Indeed, the number of communities (10), its modularity (0.293), and clustering coefficient (0.481) are relatively small, while the centrality is high (1.061). In conclusion, regarding links, the analysis of their network clearly indicates that the highly connected graph is due to a few onion domains with high in/out degree (as seen in Figures 4.4a and 4.4b). The in-degree power law of the links within the dark web is 8.588, whereas the out-degree power law is outside the 2-to-3 *scale-free networks* range — but close to it (3.234).

By comparing our work with the Surface Web structure described in previous studies [41, 191], we can observe few notable differences. First, onion domains are less prone to be linked by others, while their out-degree is smaller but similar to the one in the surface. These stats are aligned with the assumption about the undercover nature of onion domains, which explains the smaller "in" degree. We also tested whether the connections between the Tor hidden services follows the *bow-tie structure* characteristic



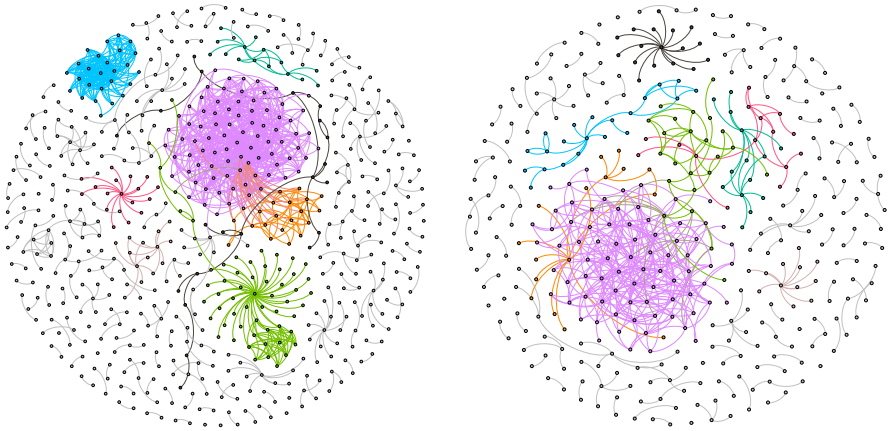
(a) Link In-Degree Distribution. The maximum in-degree has been set to 140 links or more in order to ease the understanding. Data is presented in log-scale.



(b) Link Out-Degree Distribution. The maximum in-degree has been set to 200 links or more in order to ease the understanding. Data is presented in log-scale.

Figure 4.4: Link In and Out Degree Distributions.

of the Surface Web. This structure is composed of strong connected components that are linked but not accessed by an *in* cluster, and linked to the *out* cluster. Our results indicate, that even though there is a high number of strongly connected domains, there is an absence of clearly defined *in* or *out* clusters.



(a) Resources Graph. **(b)** Redirections Graph.

Figure 4.5: Resources and Redirection Connectivity graphs plotted using the Fruchterman-Reingold force-directed layout algorithm and colored communities through modularity.

Next, we looked at the important *hubs* that are connected to a large part of the Tor hidden services graph. Two main hubs have a high number of incoming connections, and they are linked by a 13.65% and 5.29% of the nodes, respectively. The first one was a hosting service while the second was a bitcoin blockchain explorer/wallet. In the case of outgoing links, 8 onion domains linked to more than 50% of the network each domain. In particular, a single directory domain linked to 71.16% of the nodes.

Finally, resources and redirections graphs (shown in Figure 4.5) present similar structural values. The resources graph was composed of 664 nodes and 1294 edges, whereas the redirections graph contained 416 nodes and 554 edges. The out-degree power law was 4.951 in the case of resources and 5.094 in the case of redirections, while the in-degree was 4.838 for resources and 5.062 for redirections. The average shortest path for resources was smaller (2.715) than in redirections (3.596), and both of them lower than the one of the links. The diameter was smaller in this case (7 for resources and 10 for redirections).

These lower values are due to the size of these two graphs being much smaller than the links connectivity graph. However, both of them are still highly connected: 82.83% of the nodes are strongly connected in resources and 84.88% in redirections. Networks are not as sparse as in the case of links, the clustering coefficient (0.060 in the case of resources and 0.013 in

the case of redirections), and the number of communities are high (156 in the case of resources and 97 in the case of redirections), while the network centrality is small (0.038 in the case of resources and 0.017 in redirections). In these cases, as we will discuss in Section 4.3 there are serious privacy implications due to the high connectivity of onion domains. Regarding connections amid communities, resources have 28 inter-community edges while only 6 redirections. In both cases, most of these connections are between the famous onion search engine `Grams` and its bitcoin cleaner `Helix` with directories and markets.

4.3 Privacy Analysis

In the second part of our study we look at privacy implications related to the structure of the Tor hidden services and measure the web tracking prevalence and its nature within Tor hidden services. In order to measure how common web tracking is in the Dark Web as well as finding out its nature, we used the web tracking analyzer proposed by Sanchez-Rola & Santos [242] to analyze all the scripts retrieved by our crawler.

4.3.1 Dark-to-Surface Information Leakage

One of the main surprises of our structural analysis experiments is the tight connection that exists between Tor hidden services and the Surface Web. As we discussed in Section 4.2, a large number of websites contain links, import external resources, and even redirect to websites outside the onion network. While this may appear innocuous, it has important consequences for the cover nature of these websites.

Users usually visit Tor hidden services either by using a browser connected to the Tor network or by using one of the several Tor proxies (e.g., Tor2Web [8]). These services, available on the Surface Web, act as gateways to bridge incoming connections towards the Tor network. They are popular because they do not require any specific set up or installation of additional software.

When a user is using one of these proxies, instead of typing in the address bar the onion URL (e.g., `duskgytldkxiuqc6.onion`), she replaces its domain with a different one that points to the proxy service (such as `.onion.to`, `.onion.city`, `.onion.cab`, or `.onion.direct`). The proxy then acts as an intermediary to fetch the resource from the Dark Web and forward it to the user. In addition, it transparently rewrites any onion-related

URL in the returned page, to append one of the aforementioned proxied domains.

The fact that Tor proxies do not provide the same privacy guarantees of a real connection to the Tor network is well known, and even advertised on their sites. However, the main privacy issue discussed so far was that the proxy knows the user's IP address and can monitor its traffic. Hence, the user needs to trust the proxy. However, our measurement shows that from a privacy perspective, there is also another important consequence of using these proxies. In fact, in many cases not just the proxy knows the IP address of the user, but even third-parties and the target websites can get access to this information.

4.3.1.1 **Links, Resources, and Redirections**

When an onion website imports a resource from the Surface Web, its URL is not rewritten by the Tor proxy and therefore it is fetched by the user browser in the usual way, bypassing the anonymization network. This has two important consequences. First, since resources are loaded each time the website is visited, the destination domain can monitor the traffic of the source onion domain. Despite the fact that this problem is already known, we have, for the first time, measure its impact.

According to the data we collected, a handful of popular surface domains can monitor a remarkable percentage of the traffic towards onion domains. Google (13.20%), Facebook (1.03%), and Twitter (0.88%) alone cover 13.39% of the onion domains. Moreover, while these statistics are anonymous if the user is connected through the Tor network, they are not when proxies are used to access Tor hidden services. In other words, Google alone can monitor the IP address of any client who visits over 13% of the onion websites, if the user uses a Tor proxy.

The second consequence is that any onion website can simply import a library or an image from the surface web to transparently get access to every user's IP address visiting the domain using a Tor proxy. Overall, since over 21% of the onion websites import resources from the surface Web, we believe that this is a very widespread issue that should further motivate users not to use Tor proxies. Redirections from onion domains to the Surface Web are less common, but still account for a relevant percentage of the websites (2.6%). In this case, if the user is using a Tor proxy, she is redirected to the surface, thus losing completely her anonymity.

Table 4.5: Prevalence of Web Tracking Scripts in Scripts.

Type	# Scripts	% of All Scripts	Unique
Tracking	*118,675*	*44.02%*	*12,285*
– Known	22,736	8.43%	469
– Unknown Blacklisted	12,816	4.76%	1,392
– Completely Unknown	83,123	30.83%	10,438
Non tracking	*150,917*	*55.98%*	*13,053*
TOTAL	*269,592*	*100.00%*	*25,338*

4.3.1.2 **Countermeasures**

The risk of using Tor proxies is well known, but our study shows that it is even more severe than we previously thought. For users, the obvious solution is to avoid using Tor proxies and connect directly to the Tor network. Otherwise, they need to be aware that their identity can be tracked by the proxy, the target website, or by several large companies such as Google and Facebook.

Website-related privacy issues found by our structure analysis are due to the ability of a destination domain (the target of a link or imported resource) to know from which domain the request is coming and, therefore, reveal its existence to others. In order to avoid this website disclosure there are several options.

First, the developer can avoid having any external resources, links, or redirections to make impossible for an external domain to know about its existence or to monitor its traffic. If the website needs to use external resources, the developers should copy them in their website, checking that these resources do not fetch additional components from other hosts.

Second, it is possible to maintain the external connections but hide the `http_referrer` HTTP header. For example, the attribute `no-referrer` can be used to hide this property when fetching additional resources or following a link. Surprisingly, only 0.54% of the onion domains used this technique to protect their origin when importing external resources.

Table 4.6: Prevalence of Web Tracking in Onion Domains.

Type	# Domains	% Domains with Scripts	% All
Tracking	*1,992*	*76.82%*	*27.49%*
- Known	501	19.32%	6.92%
- Unknown Blacklisted	436	16.81%	6.02%
- Completely Unknown	1,886	72.73%	26.03%
Non tracking	*1,886*	*71.96%*	*25.76%*
No scripts	*4,652*	*N/A*	*64.21%*

4.3.2 Tracking

Tracking Prevalence

As mentioned in Section 4.1, we used a previously proposed tracker analysis tool [242] to analyze the scripts we retrieved from the onion websites. Using this tool, we compute the tracking prevalence with regards to every script (Table 4.5) and every onion domain (Table 4.6). The tool we use models the source code as a Vector Space Model and then it computes the cosine similarity amid the inspected script and known tracking scripts within a database. If there are no matches, it uses a machine-learning approach to categorize unknown tracking scripts.

By using this tool we can divide the tracking scripts in three categories: (i) *Known* scripts, which are at least 85% similar to already known and blacklisted tracking scripts; (ii) *Unknown Blacklisted* scripts, which were not previously known but they were imported from blacklisted tracking domains; and (iii) *Completely Unknown* scripts that were not previously known nor imported from a blacklisted domain.

According to Table 4.5, nearly half (44.02%) of the scripts present in onion domains performed diverse types of web tracking such as analytics, cookies, or device fingerprinting. This ratio is similar to what has been previously reported for the prevalence of Surface Web tracking [242].

More than 75% of the onion domains that contain at least one script, perform tracking (see Table 4.6). However, considering that 64.21% of the onion domains with HTML did not use any script, only 27.49% of all onion domains used tracking. The prevalence of unknown tracking candidates in websites was also the highest: 94.68% of tracking onion domains used at least one unknown script, while known scripts appeared in 25% of the domains, and scripts from blacklisted domains in 21%.

Table 4.7: Surface Third Party Tracking Scripts Prevalence. The percentage of scripts is computed with regards to the total number tracking scripts coming from the surface web.

Type	# Scripts	% Scripts
Surface Known	14,990	38.86%
Surface Unknown Blacklisted	12,816	33.22%
Surface Completely Unknown	10,769	27.92%
Total Surface Tracking	*38,575*	*100.00%*

Tracking Specifics

We collected 118,675 tracking scripts but, as shown by the last column in Table 4.5, only 10% of them were unique. We also checked where the tracking scripts were hosted: as a script file in the onion domain, embedded in the HTML, or in third-party domains. The majority of them was hosted in the onion domain itself: 40.39% as a separate resource and 26.11% embedded in the HTML.

Finally, to understand the tracking scripts, we performed a cluster analysis with the same methodology used in Section 4.2. In this case, we started by clustering 957 known tracking scripts using the *Affinity Propagation* algorithm [110]. Then, we computed the closest cluster for each of the tracking scripts found in our dataset. By manually analyzing the most prevalent clusters from the resulting 106 clusters, we found that 17.10% of the scripts performed statistics, 15.04% performed stateless tracking, 10.48% were used for targeted advertisement, 10.08% for web analytics, and 7.22% were stateful tracking scripts.

Hiding Techniques

We then analyzed the use of different hiding techniques, including (i) obfuscation, (ii) embedding the script into the HTML, and (iii) placing web tracking scripts in the middle of a redirection chain (i.e., in a HTML resource which is neither the source nor the origin of a multi-step redirection).

As we already discussed, our crawler uses a de-obfuscator to process each collected script file. However, to our surprise, we discovered that only a 0.61% of the tracking scripts were obfuscated.

Script embedding, which consists in copying the source code of a track-

ing script and embedding it as `<script>` in the HTML, is a common anti-tracking solutions to bypass URL-based detection schemes. In our dataset, 16.28% of the samples were embedded in the HTML. Among these samples, there are well-known web tracking scripts such as `dota.js` or `analytics.js`. For example, `dota.js`, known for performing canvas fingerprinting [13], was always embedded in the HTML. In comparison, Google's `analytics.js` was instead embedded in only 0.66% of the samples.

Finally, we identified an interesting technique in which the tracking script was hidden in intermediate URLs part of a multi-step redirection chain. For instance, a page `A` can redirect the user to `B` – which performs the tracking and then redirects again the user to a third page `C`. This setup evades those systems that load a URL and only perform the analysis on the final resources. While this technique was not widely used, a significant 1.67% of the scripts (280 unique) were found to be hosted in intermediate HTMLs.

Surface Third-party Web Tracking

Table 4.7 shows the number of scripts and its distribution between known, unknown blacklisted, and completely unknown tracking scripts. Tracking scripts coming from the surface web represented the 32.50% of all the web tracking present in the dark web and 97.04% of all the third-party tracking. Obviously, every script from a blacklisted domain was loaded from the surface web. However, the number of known web tracking imported from the surface domains is particularly high: 65% of all the already known scripts.

This is a serious issue, as adopting web tracking techniques from the Surface Web may be used to de-anonymize users. For instance, if a Tor hidden services uses the same tracking script of a site on the Surface Web, then the script can fingerprint the user even if she connects through Tor and then identify her when she connects to other websites on the Surface Web. In this case, it is important to use a browser hardened against fingerprinting, such as the Tor browser.

The vast majority of the tracking scripts imported from the Surface Web were from Google (43%) followed by Facebook (3.2%) and Twitter (1.9%). In total, we counted 146 unique surface domains. As we already discussed for imported resources, these surface domains may monitor traffic from an important number of Tor hidden services.

4.4 Related Work

Relay security [187, 50, 287, 240] and traffic analysis [208, 193, 160, 263] have been very popular lines of work regarding the security and anonymity of the Tor network. Our work focuses instead on the analysis of websites hosted in the Tor network (the so-called Dark Web).

One of the very first works that studied the nature of the Dark Web was presented by Bergman [33]. In his work, the author introduced and analyzed for the first time different characteristics of the Tor hidden services, including size, content, or their ability to remain covert.

Cyber-criminal activities in Tor hidden services had been analyzed in two recent reports by Ciancaglini et al. [55, 56], showing that this venue is commonly used to perform illicit activities by different types of criminals. Moreover, in their second report, the authors measured the distribution of several common features of the Dark Web, such as languages, market products, and criminal categories. Soska & Christin [258] presented a long-term analysis of anonymous marketplaces, providing a comprehensive understanding of their nature and their evolution over time. In a similar vein, Lewis [168, 171, 170, 169] and Intelliagg & Darksum [137] performed a number of preliminary studies of the typology of these networks and its privacy issues.

A crawling methodology for isolated Tor hidden services was also presented by Biryukov et al. [35]. But this methodology required to monitor the exit nodes of the Tor darknet, and therefore we preferred to use a less invasive approach in our work.

None of the aforementioned studies performed a complete structure analysis of the Dark Web, and neither they analyzed the privacy implications or the web tracking activity performed by onion websites. In addition, these studies had a much limited coverage of the nature of Tor hidden services, due to the fact that they draw their conclusions by accessing only the homepage of the different onion domains.

4.5 Conclusions

In this chapter we presented the first structure and privacy analysis of Tor hidden services — based on the largest experiments performed to date in this environment. We analyzed the prevalence of languages and categories, and the structure of the resultant Tor hidden services connection graph. We found that the Dark Web is highly connected but it does not exhibit the

scale-free network and *bow-tie structure* of the Surface Web.

Connections to the Surface Web from onion domains not only exist, but they are extremely common, even more than towards other onion domains. In addition, more than 20% of the onion domains imported resources from the Surface Web. In the chapter we also measure the impressive prevalence of web tracking, as nearly half of the scripts (70% of which completely unknown) were tracking and were used by nearly 30% of the onion domains.

"The digital revolution is far more significant than the invention of writing or even of printing."

Douglas Engelbart (1925–2013)

Chapter 5

Dirty Clicks: a Study of the Usability and Security Implications of Click-related Behaviors on the Web

Despite its current complexity, the World Wide Web is still, at its core, an interconnected network of hypertextual content. Over the years, static pages have been largely replaced by dynamic, stateful, web applications. However, links and other clickable elements still play a fundamental role in driving the interaction with the user: It is by clicking on links that most of the users navigate from one website to another, and it is by clicking on menus, button, and other elements of the DOM that they interact with a page and trigger JavaScript functions.

Unfortunately, browsing the web introduces important security risks. In fact, it is through malicious and compromised web pages that many computers get infected with malware and that credentials and other personal information are regularly stolen from million of users [149, 228]. On top of these criminal activities, online tracking as performed by advertisement

companies and other large corporations is one of the main privacy concern for our society [83, 243]. This translates into the fact that users need to be extremely careful when visiting webpages. For instance, it is very common to warn users not to *click* on suspicious links and to always verify the different indicators provided by their browsers to alert about potentially dangerous targets. The security community has been constantly involved in increasing user awareness and in designing and supporting user interfaces that simplify the identification of malicious links and help users avoid risky behaviors on the Web. Strangely, if on the one hand there is an obvious advantage in having "transparent" interfaces and in asking users to *judge before they click*, on the other hand it is undeniable that modern web applications have become extremely opaque from this respect. Many of them are like black boxes, where it is impossible to know what sort of behavior is triggered by clicking over a given element. Even worse, some pages even provide deceptive information, such as showing a particular link URL but then visiting a different page when the link is clicked.

In this chapter we look closely at this problem and we try to measure how widespread are these bad practices and whether they are becoming the norm rather than the exception. Most of the work performed to date on clicking behavior has been looking at the server side, i.e., on how an application can identify if a click was actually made by a real user, and not by an automated machine or a script (the so called "click fraud") [225, 192]. This is an important problem, especially in the context of the advertising pay-per-click (PPC) pricing model, but it is only one side of the picture. Instead, our study looks at the click ecosystem from the user perspective, with a focus on the different security and privacy threats that a user may be exposed to.

To fill this gap, we present an extensive analysis that sheds light on the most common click-related techniques used (intentionally or not) by web developers. Despite the fact that one may expect bad practices to be more common in dubious web sites (such as those associated to free streaming [230] or porn [289]), our experiments show that their adoption is nearly identical in highly accessed webpages listed in Alexa [252]. In particular, our results show that around 80% of the domains we tested adopts some form of misleading technique that would prevent users from making informed decisions on whether they want or not to click o a given link. Moreover, around 70% of the domains exposed users to unexpected man-in-the-middle threats, 20% of which were completely undetectable by a user even after the click was performed. Finally, possible problems related to phish-

ing are even more common, as nearly all domains opened new websites by using unprotected tabs.

The remainder of this chapter is organized as follows. In § 5.1, we will explain the unwritten *click contract*. § 5.2 details the data collections process, including domain discovery and crawling/clicking methodology. § 5.3 describes the security and privacy dangers a user is exposed by performing a click, and provides related data in each case. We then discuss possible countermeasures, scale of the study and summarize the outcome of our research in § 6.6 and § 7.4. Finally, § 9.6 discusses related work and § 9.7 concludes the chapter.

5.1 Towards a Click "Contract"

Today, there is no explicit contract among users, browsers developers, and third party website owners on what defines an acceptable behavior when a user clicks on an element of a web page. However, we believe there are a number of important *implicit* assumptions, which users often take for granted, that characterize such expected behavior.

Our hypothesis on what constitutes an acceptable behavior is based on common sense, on the way the World Wide Web was ideally designed to work, and on requirements that are often associated to usable security (such as asking users to investigate an URL before clicking on it). We can summarize these points, which form what we call the *click contract*, around two main concepts: what you see is what you get (WYSIWYG), and Trust in the Endpoints. At its core, the first concept captures the fact that links should not be deceiving and should not lie about their target URL. The second point characterizes instead the fact that a user on a website *A* clicking on a link towards a website *B* implicitly accept to receive the content delivered from those two websites (and any external element included by them). However, she may not agree to accept content delivered by other domains outside the ones mentioned above. It is important to note that, according to our definition, we do not consider background third-party content/requests (e.g., AJAX communications) a bad practice, as it is the base for many client/server interactions, and does not play a role in deceiving the user.

We can summarize our click contract in the following seven rules:
What You See Is What You Get:

1 When a user clicks on a link whose target URL is displayed by the browser at the bottom of the screen, she expect to navigate to that

same destination. In case a webpage redirections happen afterwards as a consequence of the click, the user expects that it would remain within the same domain of the displayed URL.

2 If an object is clickable but the browser does not show any domain at the bottom of the webpage, the users expect the click to generate some action within the current website and not to navigate to a different domain.

3 The user does not expect any action or user redirection when she clicks on a non-clickable element of the webpage (such as a simple text paragraph).

4 When the user clicks a HTTPS link, she expects that the communication towards the target URL will be encrypted.

Trust in the Endpoints:

5 If a user on a website *A* clicks on a link to a domain *B*, she does not expect any other domain, apart from *A* and *B* to execute code in her browser as first party.

6 If any cookie is created in the process that follows a click, the user only expect cookies from the domain she clicked or from any of the third party domains included in the target webpage.

7 If a new tab is opened by the browser after the user clicks on a link, the new tab should not be able to interact with the other opened tabs already open in the browser.

In the rest of the chapter we present a comprehensive measurement of the previously mentioned best practices, and on how widespread are violations of these seven points in the World Wide Web. We will also identify and discuss potential security and privacy threats a user may be exposed to due to the poor usability of websites that do not follow these practices.

5.2 Data Collection

The goal of our measurement is to understand how widespread among websites are bad practices related to the behavior associated to user clicks. As we will discuss later in the chapter, this is not just a usability-related

issue, but it can also have severe security and privacy implications and introduce subtle threats for web users.

To capture a view of the global *click ecosystem*, we gathered a dataset that includes top ranked domains (according to the Alexa domains list [252]) as well as domains belonging to "gray" categories offering the download and streaming of illegal content or serving adult and pornographic material. Our hypothesis is that popular website would be less deceptive with their click-related behavior, while websites associated to one of the gray categories would be more unpredictable and would introduce more risks for the end users (as confirmed by previous studies that analyzed free streaming webpages [230]).

5.2.1 **Domains Selection**

We started by populating our gray list by using different queries obtained using the auto-complete feature offered by search engines (e.g., *"game of thrones 720p mega season 7"*). In particular, we performed five different queries for each of the following eight categories: series, movies, music, games, software, TV, sport events, and adult content. To amplify the exhaustiveness of our domain retrieval phase, we executed each query in several different search engines including Google, Bing, DuckDuckGO, and Yandex and we stored the first 100 links returned in each experiment.

To avoid incurring into very popular website, we filtered the list of collected domains by removing those that also belonged to the Alexa 1K category and we performed a manual sanity-check to verify that the resulting domains indeed belonged to the categories depicted above. This brought us to a final dataset containing 6,075 unique gray domains.

Finally, in order to balance the two datasets, we randomly selected the same number of domains from the Alexa's Top 10k, Top 100k and Top1M lists (2,025 each). Combining both Alexa domains and gray domains we obtained a total of 12,150 different candidate domains for our experiments.

5.2.2 **Analysis Tool**

We implemented our click analysis tool using a custom crawler based on the well-known web browser Chrome. The crawler receives as input the main URL of a website, loads the corresponding page, and then recursively visits three randomly selected pages up to a distance of three clicks from

the home URL. This results in the analysis of 13 pages per website, mimicking a configuration already used by other researchers in similar studies [241].

It is important to remark that we consider to be "clickable" all elements that have the `cursor` property set to `pointer`. As defined by Mozilla [103]: *"The element can be interacted with by clicking on it. Used, e.g., when hovering over links. Typically an image of a hand."* Some elements have it by default, such as anchor links with `href`, others need to have it explicitly indicated or inherit it from their parent element. While it is possible from elements to react to a click even without setting a different cursor, this is per-se already a deceiving behavior. In fact, a user may decide to click on some text to select it and she would not expect this to trigger her browser to navigate to another page just because of this action. Therefore, we considered this phenomenon in Section 5.3, where we measure how many websites adopt this technique to capture unintended user clicks.

In each visited page, our crawler performed 21 different clicks. The first is executed over a randomly selected, seemingly non-clickable element with the goal of identifying websites that contain an invisible layer that intercept the user's clicks. To avoid the impact of such layer in the rest of the tests, we maintained the same session between every consecutive click on the same page.

The tool then dynamically computes the appearance of all clickable objects according to styles defined both in the CSS stylesheets and in the `style` tags embedded within the HTML. It then uses this information to rank each link according to its computed visualization size and perform one click on each of the ten largest elements. Finally, it concludes the analysis by randomly clicking on ten other clickable objects present in the webpage.

In total, this process results in up to 273 individual click for each website (21 per page). The crawler captures and records on a separate log file the entire behavior both during and after a click is performed.

As it is unfeasible to click on every single element of a page, there is a clear trade off between the accuracy of the results and the scalability of the measurement process. As a result, it is possible that some of the websites for which we did not discover any anomalous behavior were actually performing them but only on a little subset of their links. We will discuss in more details the coverage of our measurement in Section 5.2.3 and the consequences for the accuracy of our results in Section 6.6.

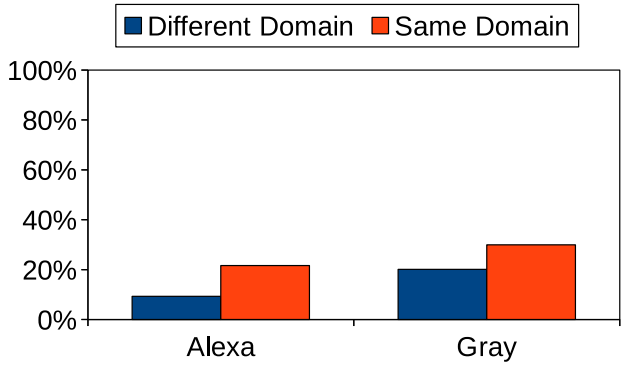
5.2.3 General Stats

Our crawler performed a total of 2,331,239 distinct clicks in 117,826 pages belonging to 10,903 different web sites – 5,455 of which belonged to the Alexa top-ranked domains and 5,448 to the gray domains, showing a balanced dataset between the two main domain categories. In order to avoid classifying websites for the specific advertisements shown at the time of crawling, we removed all the clicks linked to advertisement companies, as indicated by the list used by Firefox to block them [105]. As some advertisements may not include a domain in the `href`, we used the corresponding accesses generated after the click to make a better detection of these cases. We removed a total of 42,663 clicks following this process. Since not every domain has 13 different pages with at least 21 clickable elements each, the final number of clicks is slightly smaller than the result obtained by multiplying the individual factors. We believe our dataset is sufficient for this specific analysis, in particular given the widespread adoption of the aforementioned techniques.

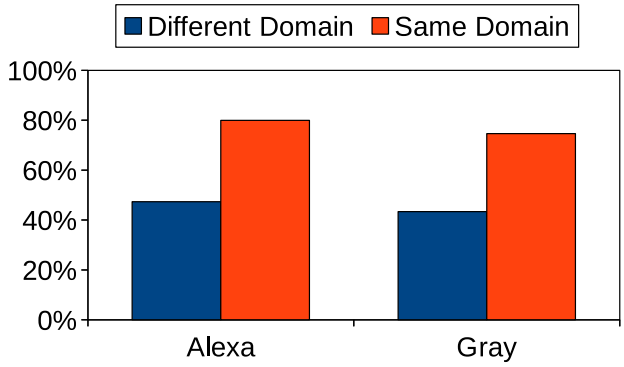
It is interesting to observe that, in average, on each website our analysis covered 28.32% of all clickable elements. From all the clicked objects, 72.33% had an `href` attribute that displayed to the user a target URL location associated to the element. The remaining 27.07% did not indicate this information, suggesting that the target resided in the same domain of the currently accessed webpage. Interestingly, only 42.19% of the links with an `href` and 45.39% of the those without used a secure transfer protocol (HTTPS).

5.3 Findings

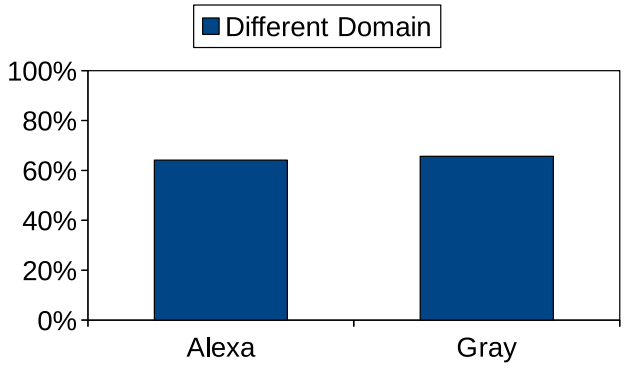
There are many security and privacy implications involved when an user clicks on an element in a webpage. In this chapter we focus on a particular aspect of those risks, namely the fact that the user has enough information to take an informed decision whether or not she wants to proceed with her action. For instance, if a user clicks a link with a `href` attribute pointing to a HTTP webpage as destination, she consciously accept the risk of sending her data in clear over the network. However, things are different when the same user clicks on a link with a `href` attribute pointing to a HTTPS URL but the web application decide instead to issue the request over the HTTP protocol. The final result remains the same (in term of communication over a cleartext channel) but in the second scenario the user had no



(a) Invisible layers



(b) Fake `href` Attributes



(c) Fake Local Clicks

Figure 5.1: Percentage of domains misleading users.

information to take an informed decision and was deceived into believing her data would be transmitted over a secure channel.

In this section we present threats that the users could not predict before clicking, as they are much more dangerous and difficult to protect even for

Table 5.1: Occurrences of webpages misleading users.

Type	Total Occurrences	Targeting Different Domains
Invisible Layer	19,696	54.33%
Fake `href` attributes	138,860	31.14%
Fake local clicks	123,959	100.00%
TOTAL	282,515	63.00%

experienced users with a security background due to the lack of information required to perform any preventive actions.

5.3.1 Misleading Targets

One of the most important aspects for the user when performing any type of click in a webpage, is trust. Trust implies that when the webpage explicitly mention the target URL, this is indeed where the browser will navigate to. Even though many users take this trust for granted, webpages do not always follow this rule and often mislead users into performing actions that are different from the intended ones. In our study, we have detected three different types of misleading clicks:

- **Invisible Layer:** The user clicks some non-clickable object of the webpage (e.g., some random text or image). Despite the fact that there should not be any expected result from this specific action, on some websites this may triggers a webpage redirection or the opening of a new tab.
- **Fake `href` Attributes:** In this case, the user wants to click on a given element, such as a simple <a> tag. The user's expectation is that the browser will go to the website indicated by the link (as specified in the `href` attribute). However, the user is instead forced to visit a different website.
- **Fake Local Clicks:** When some object in a webpage is clickable but the browser does not explicitly indicate a target URL, the user may reasonably expect the destination to be in the same domain of the current website. Nevertheless, this statement is not always true and users are sometimes redirected to completely unrelated domains without any prior notice.

Results

As shown in Figure 5.1a, roughly 20% of the Alexa website and 30% of the gray dataset contained an invisible layer that captured the user's clicks. Moreover, while webpages in the Alexa dataset had a 50/50 chance of redirecting the user to a completely different domain, this percentage grew to 2/3 of the clicks on the gray dataset. If we check the global numbers (Table 5.1), we can see that more than 50% of all the redirections/new tab opens using this technique were performed to a different domain. Our data shows that this is a very widespread problem and that in the majority of the cases the target URL is not even located on the same domain.

Figure 5.1b shows that the vast majority of websites (between 70% and 80%) mislead users by reporting incorrect `href` attributes on some of their links. Even worse, in over 30% of the cases those links pointed to completely different domains from those reported in the displayed URL.

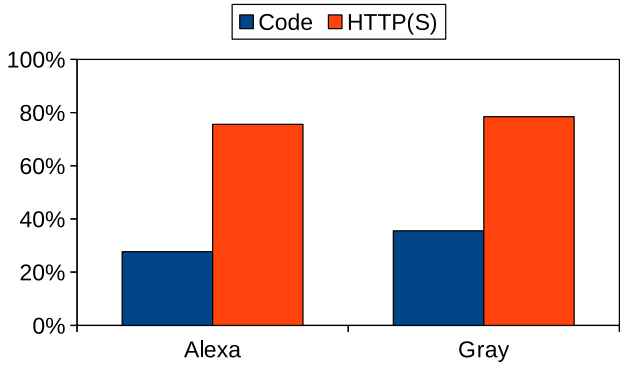
Finally, fake local clicks are also quite common on the web with around 65% of the websites we tested (Figure 5.1c) adopting this technique. Interestingly, the total number of occurrences is the same as the fake `href` attributes, showing a similar global trend between both techniques (Table 5.1).

To sum up, misleading targets are worryingly popular among all types of websites. In fact, despite the common intuition that this type of techniques would be prevalently used in gray webpages for aggressive advertisement reasons, our results show that most of these bad practices are equally common in both datasets.

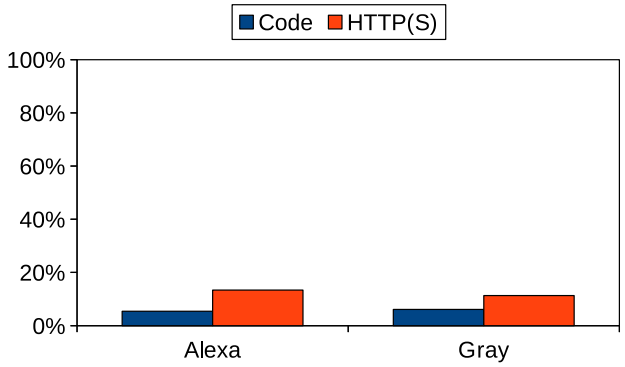
5.3.2 Users Redirection

Even when a click initially behaves as expected, it is still possible for the user to be redirected to different pages without her consent. Of course, redirections are very common on the Web and can be used for perfectly legitimate reasons. Moreover, if a web page `a.com` contains a link to `b.com` which will eventually redirect to another domain, the owner of `a.com` has no control over this behavior. Nevertheless, we decided to measure and report how prevalent this behavior is as, from a user point of view, it still results in hiding the final target of a click.
Ignoring internal (i.e., to the same website) redirections, we can classify the remaining in two broad families:

- **Different Domain:** This family includes all the redirections to do-



(a) Different Domain



(b) Hidden Domain

Figure 5.2: Percentage of domains redirecting users.

mains different from what the user was expecting when performing the click. For example, if the user clicks a link in `a.com` pointing to `b.com`, any redirection involving any of the two domains is considered legitimate. This is the case in which `b.com` uses a redirection to point to another URL in the same website. However, if the users clicks on a link to `b.com` and ends up on `c.com`, this can potentially be deceiving.

- **Hidden Domain:** This is a more severe variation of the scenario described above. In this case, the user clicks on a link pointing to `b`, which *temporarily* redirects to `c`, which then in turn immediately redirects back to `b` – thus introducing a third domain in the redirection chain that the user would not even be aware of (as the browser would likely not show this intermediate step).

On top of these two classes, there is another orthogonal classification related to the specific method used to perform the redirection. On the

Table 5.2: Occurrences of webpages redirecting users.

Type	Total Occurrences	HTTP(S)	Code
Different Domain	525,975	68.68%	31.32%
Hidden Domain	42,558	31.31%	68.69%
TOTAL	568,533	65.88%	34.12%

one hand, we have the HTTP(S) redirection, where the request can include the `Set-Cookie` header to create different cookies in the user's browser for that specific domain. The HTTP code employed in these redirection is `30X`, where the last number specifies the reason for the redirections (e.g., `301` is used to notify that the requested resource has been *Moved Permanently*). On the other hand, we have code-based redirection that do not happen by means of a HTTP request, but by code being executed in the webpage, once parsed and loaded by the browser. The problem in this type of redirection is that it gives complete uncontrolled code execution rights to the domains involved. They rely on HTML refresh using a `meta` element with the `http-equiv` parameter or directly with JavaScript using `window.location` or any other equivalent method.

Independently from the method used to redirect the browser, for our study, we are particularly interested in how transparent it is for the user which domains have been visited during the transition, in particular in the case of multiple consecutive redirections.

Results

As shown in Figure 5.2a, 75%-80% of the all domains perform HTTP(S) redirections pointing to completely different domains of the ones expected by the users. Regarding code redirections to different domains, an impressively 35%-40% of them use this technique. This is particularly worrying because of the aforementioned security problems that derive in possible uncontrolled code executions or cookies. The user was never notified that she was going to give these rights to those domains. Percentages are consistent among both Alexa top-ranked websites and gray websites. According to the global occurrence data presented in Table 5.2, the percentages follow a similar trend, with a majority of domains redirecting through HTTP(S) and a not negligible one third of domains allowing code execution.

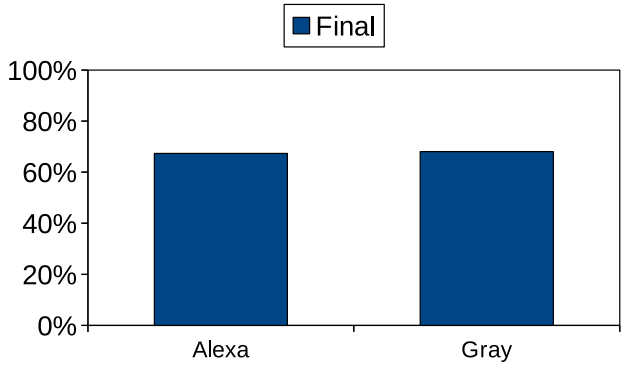
More worryingly, around 15% of the analyzed domains stealthily allows other domains to gain uncontrolled cookie or code execution rights, including them in the middle of redirections chains that end in the correct domain. Nearly 10% of them actually allow intermediate hidden domains to execute code without any control. Checking the total occurrence numbers (see Table 5.2), this percentage is much bigger, with nearly 70% of the occurrences the ones that allow hidden domains to execute their own code. The problem here is very important, as all the hidden domains (not detectable for the user) that are using code redirections have total uncontrolled rights to execute anything they want in the user's browser (e.g., tracking and profiling the user). And the user was never informed that she was going to give these rights to those domains.

5.3.3 Insecure Communication

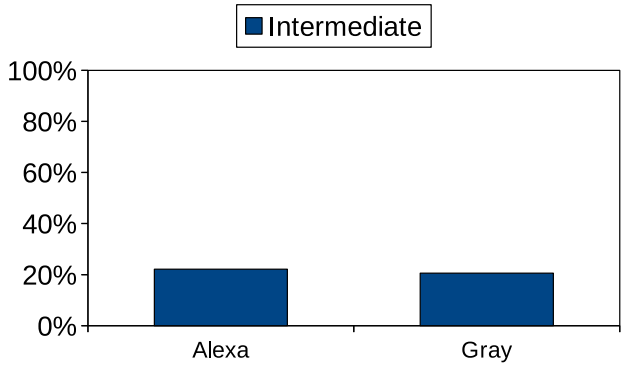
Man-in-the-middle attacks to violate the users privacy, steal credentials, or even inject/modify the data in transit are a serious threat to web users [51, 291]. When a user visits a website over HTTP she implicitly accepts the fact that her traffic would not be protected against eavesdropping. However, when a user clicks on a link that displays an HTTPS URL, she only agrees to send her data over a protected channel.

Unfortunately, in reality we found that this behavior is rarely the rule. In particular, we identified three main scenarios in which this requirement is not met:

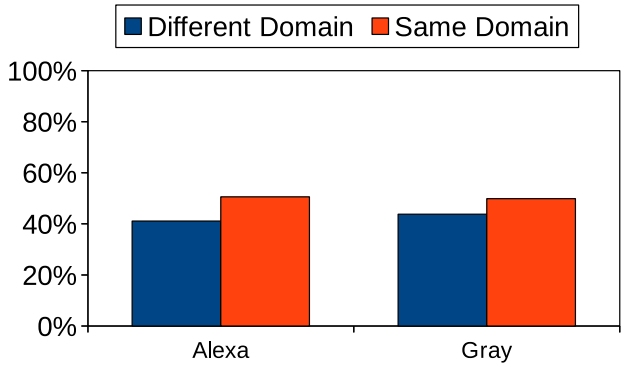
- **Insecure Access:** This is the basic case in which the user clicks an element pointing to a HTTPS URL but eventually the browser (either from the beginning of because of a redirection) drops the secure channel and ends up visiting a page over an insecure HTTP connection.
- **Hidden HTTP Connection:** In this very subtle scenario, the user initially clicks on an HTTPS URL and eventually lands on a website served over HTTPS. Everything may therefore seems normal, but unfortunately there were intermediate HTTP webpages (invisible to the user) visited by the browser before reaching the final destination. In other words, the two endpoints are secure but the entire communication was not – without the user being aware of it.
- **Unexpected Mixed Content:** By default, over a secure connection, browsers block what is generally known as active mixed content, i.e.,



(a) Insecure Access (HTTP)



(b) Hidden HTTP Connection



(c) Unexpected Mixed content

Figure 5.3: Percentage of domains creating man-in-the-middle threats.

elements served over HTTP that can directly interact with the content of the page. However, other elements such as images and video files (i.e., passive mixed content) are allowed. This opens the door to possible security and privacy attacks that use passive mixed content.

Table 5.3: Occurrences of webpages creating MitM threats.

Type	Total Occurrences	Unique Domains
Insecure Access	185,984	23,570
Hidden HTTP Connection	43,773	7,292
Unexpected Mixed Content	279,550	22,322
* Different Domain	194,019	17,093
TOTAL	465,534	45,892

For instance, an element loaded via HTTP can be modified to a 401 *Unauthorized* response that includes a `WWW-Authenticate` header asking for a confirmation of their credentials (which will be sent directly to the attacker) [11]. Following our usual guidelines, we only measure mixed content loaded in webpages from domains different from those the user was aware of contacting.

Results: Figure 5.3a shows that approximately 70% of all the domains we tested contained at least one link in which they insecurely redirected users over an HTTP connection when they explicitly specified HTTPS in the destination URL. To make thing worse (see Figure 5.3b), a non-negligible 25% of these insecure redirections happen in the middle of theoretically secure connections, making it impossible for the end user to detect this dangerous behavior. Overall (see Table 5.3), 23,570 unique domains were involved in these task (sum of unique domains per accessed domain) and 30.94% of them were related to intermediate undetectable insecure HTTP connections.

Regarding the non-informed mixed content fetched from third-party webpages, we measured that around 40% of all the domains have at least one in their redirection chains (see Figure 5.3c). In fact, only 5% of the domains include mixed content only from the same domain — the one that is expected and accepted by the user. This shows that more than half of the domains indirectly put their users in jeopardy not by performing an insecure redirections, but by loading external content over an insecure channel. Furthermore, if we count the unique domains that suffer from this problem, from a total of 22,322 different domain, a remarkable 76.57% belong to completely different domains of those expected by the user (as shown in Table 5.3).

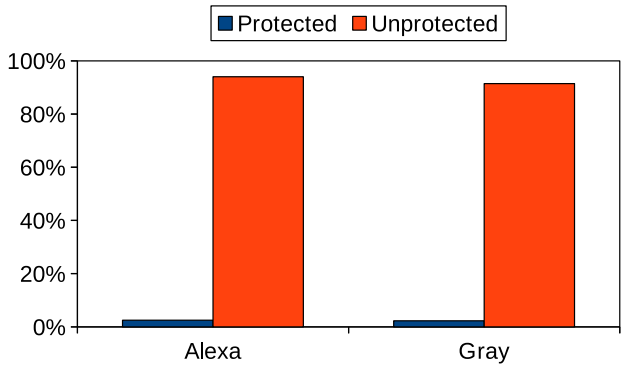


Figure 5.4: Percentage of domains opening new tabs.

5.3.4 Phishing Threats

While phishing attacks are usually associated to spam or scam campaigns, it is also possible for users to encounter a phishing website when surfing the Web. In this section, we explore how many websites are jeopardizing their visitors through their poor links hygiene. In fact, when a website opens a new browser tab or a new window, this new page obtain a reference to the original website that has triggered its opening through the `window.opener` object. To prevent the new site to tamper with the content of its parent, modern browsers are equipped with blocking capabilities through specific cross-origin actions derived from the well-known same-origin policy. However, it is still possible for the new tab to redirect the original opener website using the `window.opener.location` object, thus bypassing this protection [106].

In this way, from a newly opened tab, a miscreant is capable of detecting the domain of the opening website, and then redirecting it to a phishing website of that same domain (maybe adopting some typosquatting techniques [267, 197] to make it harder for the user to notice the replacement), and finally even closing the new tab. For example, a user on Facebook can click a link to an external website which could act perfectly benign except from replacing the Facebook page itself with a fake copy that may be used to phish users into disclosing personal information or login credentials. This makes the scheme very difficult to detect also for an expert user. This type of attack is popular named as tabnabbing [221, 232].

A simple solution exists to protect against this type of attacks: when a website includes links to external resources, it can specify `rel="noopener noreferrer"` to prevent the new page from redirecting the parent URL to

Table 5.4: Occurrences of webpages opening new tabs.

Type	Total Occurrences
Link (`_blank`)	239,628
JavaScript (`window.open`)	613,457
TOTAL	853,085
* Protected	1,324

another domain [121, 45]. Equivalently, when a new tab is opened via JavaScript, setting the new window's opener to `null` solves the problem. However, still today many webpages do not adopt any protection methods when opening new tabs, exposing themselves and their visitors to these phishing attacks.

Results

During our experiments, a stunning 90% of the websites opened at least one new tab as a result of a click on one of its elements (Figure 5.4). Overall, this accounted for 853,085 new tabs. As reported in Table 5.4, the majority of them (71.91%) were opened by using a piece of JavaScript code.

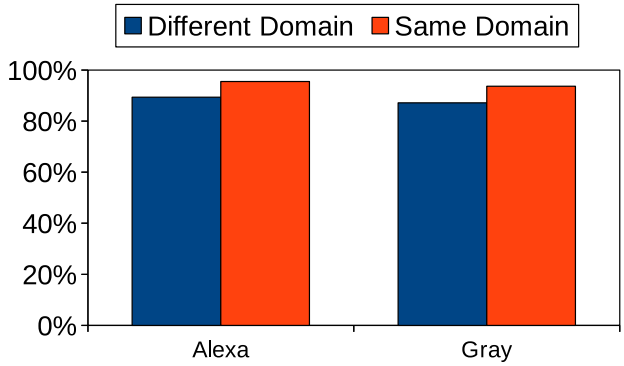
Despite this behavior is extremely widespread, we found that only 2% of the examined domains employed prevention techniques to secure their users from potential phishing attacks. Globally (see Table 5.4), the number is even smaller with only 1,324 protected links our of more than 850K visited ones.

To sum up, these results show that nearly all of the new tabs opened are completely unprotected from possible phishing attacks. Moreover, opening new tabs is an highly widespread action that most webpages do at some point.

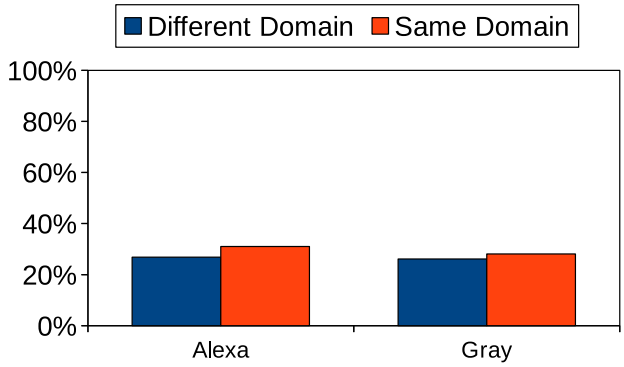
5.3.5 User Tracking

One of the biggest concern nowadays regarding web privacy, is web tracking. Web tracking is defined as the possibility of obtaining and/or inferring the users' browsing history, or identifying the same user in multiple different accesses.

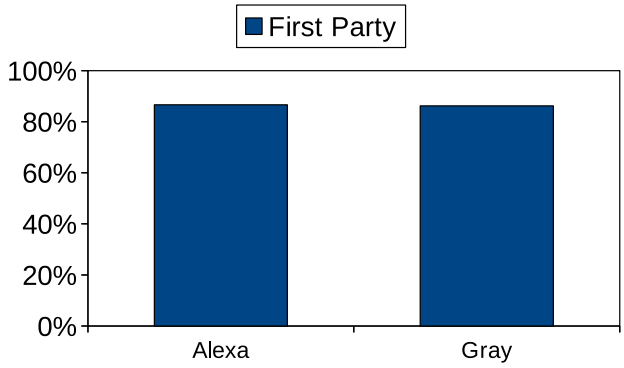
The first and still most widespread method for web tracking is based on cookies. In its most basic form, when a user visits a website `a.com`, she ac-



(a) Undesired Cookies (All)



(b) Undesired HTTP Cookies



(c) First-Party Control Bypass

Figure 5.5: Percentage of domains creating tracking threats.

knowledges that several cookies can be created and stored in her computer. These cookies can be set from the website she is visiting (`a.com`) or from a third-party domain (`z.com`) that may be also present in other websites (e.g., `b.com`, `c.com`, and `d.com`). This allows `z.com` to follow the user activity if

she also visits these webpages. Libert recently probed [172] that in most cases, the main domain does not notify the user about those third-party domains creating cookies. Nevertheless, this specific cases are out of topic, so we take an optimistic position and we consider those cases as benign.

What we are instead interested in measuring is the fact that the user is not even aware of additional cookie generations in the following cases:

- **Undesired Cookies:** If a user clicks an object pointing to `a.com`, she does not expect any other cookies besides the ones created by `a.com` and its direct third parties. Thereby, we will consider as undesired any cookie that does not follow this simple rule. For example, imagine that the previous click redirects you to `b.com` and later, though JavaScript, to `c.com`. All cookies set by `b.com`, `c.com`, and their respective third parties would be considered as undesired cookies.

- **Undesired HTTP Cookies:** In several cases, the problem is bigger than just having a large number of undesired cookies created in the browser. Sometimes, these cookies besides being undesired, they are also insecure (even if the user clicked a link directing to a secure webpage). For instance, a miscreant can perform a man-in-the-middle attack and steal those cookies or even modify them to allow future attacks or perform tracking to this user.

- **First-Party Bypass:** Browsers offer three different options to control the type of cookies they accept: (i) accept all cookies, (ii) disable all cookies and (iii) disable third-party cookies. However, users complained that these options were too coarse-grained to match the actual users' needs and therefore some browsers (e.g., Firefox and Safari, and presumably the rest in the future) introduced a fourth option [104, 22]: accept cookies from the domain the user is currently visiting, but only allows third-party cookies from webpages previously visited by the user. Nevertheless, the current click ecosystem undermines this option, as the user end up *unintentionally* visiting many domains – which will therefore be whitelisted and allowed to set cookies.

Results

In our experiments we did not count the number of cookies, but the number of domains that created undesired cookies. For example, if `b.com` created 5 undesired cookies and `c.com` 3 undesired cookies, we would report

Table 5.5: Occurrences of webpages creating tracking threats.

Type	Total Occurrences	Unique Domains
Undesired Cookies	1,924,371	188,992
* Different Domain	1,241,806	165,735
Undesired HTTP Cookies	80,494	19,338
* Different Domain	73,171	18,175
First-Party Bypass	500,073	104,075
TOTAL	2,504,938	312,405

2 (`b.com` and `c.com`) in our statistics (see Table 5.5). Moreover, unique domains are counted as the sum of unique domains per accessed domains.

Impressively, around 95% of all domains created undesired cookies (see Figure 5.5). Globally, 64.53% of all occurrences were created by different domains, making a total of 188,992 unique domains.

Analyzing the specific case of insecure undesired HTTP cookies, the number are much lower, but still concerning due to the security and privacy problems they incur. 30% of domains created these type on dangerous undesired cookies and our data shows that 90.90% of all the occurrences were performed by different domains (18,175 unique ones).

Finally, we found 500,073 occurrences of unspecified domains becoming first-party webpages (104,075 unique) and thereby bypassing the most recent cookie control policy implemented in browsers. Figure 5.5c shows that 87% of the websites (both in the Alexa and Gray categories), once visited by a user, as a side effect result in a new domain being added to the whitelist. As these domains were not visible to the user at any point in time before the click (and often even after), the user is completely unaware that they would be from now on considered as "visited webpages" for her browser.

5.4 Discussion

5.4.1 Impact

Browsing the Web, and therefore clicking on links and other elements of a page, is one of the most common activity that billions of users perform every day over the Internet. To avoid potential risks for the security and

Table 5.6: Security and privacy danger levels related to the different click implication categories.

Category	Type	Risk
Misleading	Invisible Layer	High
	Fake `href` Attributes	Medium
	Fake Local Clicks	High
Redirecting	Different Domain	High
	Hidden Domain	Very High
Communication	Insecure Access	Medium
	Hidden HTTP Connection	Very High
	Unexp. Mixed Content	Low
Phishing	Unprotected New Tabs	High
Tracking	Undesired Cookies	Medium
	Undesired HTTP Cookies	High
	First-Party Bypass	High

privacy of users, it is imperative that web site developers respect some basic rules – dictated by common sense – on the consequences and on the behavior of clickable elements.

Our measurements show that violations to these rules are not isolated corner cases, but instead an extremely widespread phenomenon that affects every type of web application. In Table 5.6 we tried to summarize the security and privacy risks and their corresponding possible scenarios. We take the worst possible scenario as a baseline in each case. For instance, in redirections, we do not consider HTTP(S) redirections, but only code-based ones. Moreover, we assessed the risk on the expectation of the user when performing the click.

We will now discuss the precision of the results obtained in this study and we will also propose different approaches to avoid these type of security and privacy problems for the end user.

5.4.2 Precision

Following the click analysis structure presented in § 5.2, we performed roughly 2.3M different clicks. If we calculate the percentage of clicks we made comparing to all the possible clicks in each domain and compute

the mean, we obtain 28.32% – which means than in average we clicked one third of the clickable elements in the pages we visited. We also calculated the percentages of clicks per domain that were affected by the various problems we identified in § 5.3 and computed the values corresponding to different quartiles (e.g., Q3 and Q4) to obtain a general overview.

With the data relative to the quartiles and the percentage of total clicks performed, we can statistically estimate the probability of detecting at least one case of every dangerous category with the amount of clicks we performed in a given website. In fact, we can model a website as a urn containing links of two categories: those affected by a given problem X and those that are not. Since we can estimate the percentage of the two types of links based on the data we collected, and we know that for a certain website we randomly visit (i.e., extract from the urn) a certain number of elements over the total number contained in the urn, we can estimate the odds of picking at least one link affected by the problem [32]. We repeated this computation for all the types of problems discussed in the chapter. In average, the probability of misclassifying a website just because we did not test the right link varied from 0% (for tracking-related threats) to 4.7% in the case of insecure communications. These values show that when a website suffers from a poor behavior related to its links, this often affects a large percentage of its elements, thus making our sampling rate of testing one out of three links appropriate to estimate the presence of the different problems with a high accuracy.

5.5 Countermeasures

During our measurement, we identified several bad practices on how click-related deliberately introduced by the developers (e.g., to avoid cookie control policies), we believe that the main cause is a lack of awareness, of clear guidelines, and a poor understanding of the problem and the risk it may introduce.

In order to avoid this last situation, we will present some simple best practices for web developers.

5.5.1 Awareness

We hope that this chapter can raise awareness about the widespread adoption of misleading links and potentially dangerous click-related behaviors.

Results

https://www.theguardian.com

✔ means secure and ✘ means insecure

Category	Type	Risk
Misleading	Invisible Layers	✔
	Fake href Attributes	✔
	Fake Local Clicks	✔
Redirecting	Different Domain	✔
	Hidden Domain	✔
Communication	Insecure Access	✔
	Hidden HTTP	✔
	Unexp. Mixed Content	✔
Phishing	Unprotected New Tabs	✘
Tracking	Undesired Cookies	✘
	Undesired HTTP Cookies	✔
	First-Party Bypass	✔

Figure 5.6: Screenshot of the resuilts shown by our PoC service.

To make our work more usable for end users and developers alike, we decided to implement our checks in a free service that can test a given web page and generate a report describing the bad practices identified in its clickable elements.

We believe that such a tool can be extremely useful for end-users interested in validating suspicious websites before visiting them, and in particular for web application developers to discover how they could improve (see next Section for more information on best practices) both the usability and the security of their website.

A proof-of-concept demo of the service is available and a screenshot is

reported in Figure 5.6 (`https://clickbehavior.github.io`).

5.5.2 Developer Guidelines

Our online service also provides guidelines to help developers avoid common mistakes. The idea is to guide developers to implement web pages that adhere to the *click contract* described in § 5.1. In particular, to increase the users trust and reduce risks, we believe websites should always strive to adopt the following rules:

- **URL Transparency:** When an element declares a target URL, a click on that elements should navigate the user only to that target (or, through redirections, to other pages in the same domain of the target).

- **Safe Default:** It is acceptable, and even needed in some situations, for clickable elements not to show their target URL. In this case, as no new domain is shown to the user, the clickable object should never direct users to external domains, but only to other web page that are part of the same site.

- **Avoid Hidden Steps:** When redirections are necessary, the set of visited domain should be visible to the user. In particular, if a user on `a.com` clicks on a link to `b.com`, the navigation chain `a -> c -> b` (with `c.com` visited "behind the scene" before reaching the original target) is not acceptable.

- **Explicit Channel Security:** Users should always be aware when they transition from a secure to a non-secure connection. When they click over a HTTPS link, they expect all the following communications to be encrypted and webpages should try to honor this expectation. The most important mistake to avoid is the one where a "hidden" HTTP connections happens in the middle of different HTTPS redirections, as the user may think that all the intermediate redirections where secure even if they were not. Mixed content (secure webpage loading insecure elements) should also be limited to only extreme cases.

- **Tracking Control:** In order to allow the user to decide what cookies should be stored in her browser, websites should not redirect connections to different domains without previous consent. These domains

can in fact create cookies or, even if they do not, they are automatically whitelisted to allow them to set cookies in the future . More importantly, avoid sites that create cookies in a insecure way when the user clicked over an HTTPS link .

- **Source Protection:** Open new tabs in HTML by including the additional`rel="noopener noreferrer"`. In case the tab is open from JavaScript, developers should set the newly created window object's `opener` to `null`.

5.5.3 Client-Side Countermeasures

As we cannot expect all web pages to follow the aforementioned guidelines, either because the developers may decide otherwise (e.g., in case of malicious actors) or because the they might not know how to follow them, it is important to have a second line of defense to protect the users. Luckily, a browser-based countermeasure could easily prevent most of the dangerous side effects by following these simple rules:

1 When an element does not display any `href`, only allow local URL targets and redirections. When a domain is instead displayed, only allow access to that domain. Any violations of this rules should result in a warning that informs the user about the real target the browser is going to navigate to.

2 If the user clicks a HTTPS link, only allow secure connections and either block or inform the user about any HTTP request. In particular, it is important for the user to be aware of any intermediary "hidden" insecure connections that may be performed before reaching again an HTTPS URL.

3 When opening a new tab, either through a link or a JavaScript code, disable the `opener` object by default. If the website needs to access or modify the opener's webpage, it should explicitly asks for that permission beforehand.

We implemented a browser extension providing the first two countermeasures and a transitory solution for the third one. While we believe the third point suggest a much more secure default behavior for new tabs, its

implementation would require support from (at least some) website developers and therefore is not under our control. As a temporary option, our extension will block all control of the `opener` object from different domains and will only allow this type of control between webpages from the same domain.

A preliminary version of our extension is available at `https://click-behavior.github.io`.

5.6 Related Work

Researchers have looked at different ways users are **misleaded** to perform actions they did not originally intend to perform [66]. For instance, researches from Google analyzed the distribution of fake anti-virus products on the Web [231]. They detected over 11k domains involved in such user-misleading actions. More specific to user clicks, Fratantonio et al. [108] proposed the cloak and danger attack on Android devices. In this attack, user are fooled into clicking elements while actually clicking on others, resulting in a full system compromise. Many other works analyzed the specific case of clickjacking[236, 28, 135], where a malicious website tricks the user into clicking on an element of a completely different website without realizing, for example, making it invisible or just partially visible.

Redirections are often used for legitimate purposes (e.g., to redirect users from a temporally moved webpage), but other times are abused by attacker for malicious reasons. For example, Lu et al. [176] were able to classify different search poisoning campaigns by checking their redirection chains. Stringhini et al. [262] proposed a similar idea to detect malicious webpages. By aggregating the corresponding redirection chains and analyzing the characteristics of those graphs, the authors were able to identify scam pages or webpages that deliver malicious software. Our work differs in many ways as we do not try to use redirection to perform any classification. Instead, we check what are the possible risks a user may suffer because of obfuscated redirection chains.

The problem of possible **man-in-the-middle** attacks have been extensively analyzed in the web. Chang et al. [51] screened the integrity and consistency of secure redirections that happen when accessing the main page and login page of domains listed in Alexa. The authos show that the majority of the main pages are vulnerable to attacks. Later, researchers from Google, Cisco, and Mozilla measured the adoption of HTTPS on the

web [91]. They conclude that globally most of the browsing activity is secure. East Asian countries, on the contrary, exhibit lower percentage of adoption. Regarding mixed content, Chen et al. [52] investigated the dangers of this type of insecure content. In their paper, they show that many webpages failed the migration to HTTPS and still included a large number of vulnerable HTTP content. None of the aforementioned studies analyzed how this security and privacy problems is related to the click ecosystem.

Phishing attacks have often been associated to spam emails [127, 93]. Therefore, the majority of the effort to stop these kind of practices was in the early detection of malicious emails [296] or on the detection of phishing pages on the Web [295, 177, 111]. However, we are not aware of any study that tries to identify how common are phishing threats created by insecurely opening new tabs. Our works shows that nearly all the targets opened, either via HTML or directly through JavaScript, suffer from this problem. Even if defenses exist for both cases, they are unfortunately very rarely implemented.

User **tracking** is a increasingly growing concern that has attracted a considerable amount of attention from researchers and end users [235]. Lerner et al. [167] studied the evolution of tracking over the last 20 years, showing an impressive growth in adoption and complexity. More recently, Sivakorn et al. [255] studied the case of HTTP cookies and the corresponding exposure of private information. They showed that various major services were suffering from this kind of problems. On the other hand, we analyzed the concept of undesired cookies that are the consequence of user clicks and we measured how many of those are insecure. Furthermore, we also discovered that due to the nature of the current click ecosystem, many domains are bypassing the most modern cookies control policies, making them completely useless.

5.7 Conclusions

Using the mouse to click on links and other interactive elements represents the core interaction model of the Web. Even though there have been several studies about click frauds and other techniques to deceive users, no comprehensive study of the click ecosystem has been performed to date. Due the possible security and privacy attacks that users may suffer when clicking on webpages, we perform in this work the first measurement of click-related behaviors and their associated consequences.

We first identified different types of undesired actions that may be triggered when a user clicks on, in principle, harmless elements. We classified them in five different categories in relation to their technique and/or security and privacy threat. In order to assess how widespread are these behaviors on the Internet, we then implemented a crawler we used to perform nearly 2.5M clicks on different types of domains of various popularity.

Our results show that these dangerous situations are extremely common in all types of domains, making a relevant number of users vulnerable to many different attacks. In order to solve this problems, we proposed a list of best practices for web developers and we implemented a service that can check their compliance in a target web page. Moreover, as users cannot simply trust developers to do the right thing, we also proposed a number of client-side countermeasures that we implemented in a browser extension that can transparently protect against the aforementioned dangers.

"People often claim to hunger for truth, but seldom like the taste when it's served up."

George R.R. Martin (1948–)

Chapter 6

Can I Opt Out Yet? GDPR and the Global Illusion of Cookie Control

On May 25th, 2018 the European Union's (EU) General Privacy Data Regulation (GDPR) entered into effect. This resulted in users worldwide being flooded with emails about updated privacy terms, and in many websites starting to ask for explicit consent to collect and share user information; some even redirected users to a text-only version of their site or refused to serve content to anyone connecting from the European Union [131]. The reason for this is that the GDPR introduced a new stern regulation on user data, forbidding much of the tracking previously performed in more than 90% of the highest-traffic websites [83, 243].

Much of the economy behind the Web is moved by advertising: a huge, fast-growing industry estimated to be worth around 227 billion dollars in 2018 [195]. A cornerstone of the digital advertisement industry is personalization in the form of targeted ads, which increase the likelihood that users will follow them. Unfortunately, collecting the very data which is needed to personalize advertisements carries important privacy risks. In particular, tracking done through web cookies results in advertisers having access to a large part of user's browsing history—a particularly sensitive piece of

data, since it can result in disclosing all kinds of confidential information (e.g., medical conditions or political opinions).

In order to protect its citizens' privacy, the European Union introduced a set of laws. For instance, in 2009 the ePrivacy directive [86] required user consent before websites could track them. However, the implementation of this directive often resulted in simple pop-up messages that did not provide any real option to users. Moreover, its adoption was limited by the fact that it was implemented differently in the various EU countries, often raising criticisms [164]. The limitations of this legislation were one of the motivators for the GDPR [224], which was approved in April 2016 and entered in force on May 25, 2018 (we discuss in more details the content of the GDPR in §7.1). Just a month after that, California also introduced a similar law, which will take effect in 2020 [163].

Researchers already started to investigate the effect of the GDPR in two independent studies. On the one hand, *whotracks.me* [286] collected data on which tracking is performed when users visited different websites, and observed that the advertising market is recently shifting towards fewer larger companies. On the other hand, Degeleing et al. [68] looked at how European websites have changed after the GDPR and noted an increment in the amount of information and control available to users in the EU. While these two studies show that the GDPR is already changing the tracking panorama, it is still unclear what the final effect is for the end users and whether, and to which extent, it is now possible for European citizens to effectively opt-out from tracking. Moreover, it would be also interesting to understand the actual "boundaries" of the effects of this regulation and whether the required privacy controls are different between the EU and the rest of the world.

To answer the previous questions for cookie-based tracking, we performed an extensive manual analysis of 2,000 popular websites, located in different countries and belonging to several categories. In particular, we visited each site and tried to opt out from tracking every time we could. In the meanwhile, a custom browser plugin collected all cookies set by those websites (both before and after our opt-out attempts), as well as the information presented to the user and the privacy policies and the privacy controls available for expressing an informed consent.

Our results show that tracking is prevalent, happens mostly without user's consent, and opt-out is difficult. Most websites perform some form of tracking, and 92% of them do it before providing any notice to the user. Only 4% of the websites provide a clear opt-out option in their cookie no-

tice, but even when they do opt-out is mostly ineffective—only 2.5% of the websites erase some cookies.

The impact of the new regulation is largely perceivable globally: in particular, USA-based websites have a behavior which is on aggregate similar to the one of EU-based ones. While sites in other countries appear to follow the regulation less, they are still often impacted indirectly: opting out of tracking through third parties reduces the amount of tracking. Within each website category we observe a similar degree of tracking, with a few exceptions that deviate from the typical behavior of commercial websites.

The remainder of this chapter is organized as follows. §7.1 gives an introduction on HTTP cookies and the GDPR. §6.2 describes the methodology used in the experiments and data analysis. §6.3 details the results obtained in the large-scale evaluation, describing general findings, and differences between regions and categories. §6.4 analyzes the complexity of cookie and privacy policies. §6.5 discusses a few in-depth case-studies for the tracking behavior of some websites. §6.6 elaborates on the results and discusses their implications. §6.7 presents related work with respect to user tracking and the analysis of policies. Finally, §6.8 concludes the study.

6.1 Background

In this section we provide a short introduction to HTTP cookies and their relation with the GDPR. For a more in-depth discussion on the topic, we refer the reader to [4].

6.1.1 HTTP Cookies

Cookies [196] were first introduced by Netscape in 1994 to enable stateful browsing over the stateless HTTP protocol. They allow small pieces of data created by a web application to be stored on the client-side on the web browsers. For example, they can be used to store information on user authentication, preferences, and to keep track of session identifiers that applications need to enforce complex workflows (e.g., "shopping carts"). Each cookie is characterized by a name, a value, a URL it is associated with, and an expiration date. Whenever a browser issues an HTTP(S) request, all cookies mapped to a configurable prefix of the target URL are sent to the corresponding web server.

On top of their original goal of providing stateful navigation, nowadays cookies are routinely used also to track users *across* different websites,

most often for advertising and analytics [83, 243]. This is possible because websites often include resources provided by third-party domains such as advertisement companies. Upon connection to the website, these third-parties can set uniquely identifiable cookies on the visitor's computer; they will then be able to track these identifiers across multiple associated websites, thus reconstructing part of the user's online activity.

While cookies are the most widespread way of tracking users on the web, they are not the only one. Work on other kinds of user tracking, which is outside the scope of this paper, is discussed in §6.7.

6.1.2 The GDPR

The GDPR [224] is an EU legislation (in effect since May 25, 2018) that regulates the handling of personal data with the goal to *"strengthen individuals' fundamental rights in the digital age and facilitate business by clarifying rules for companies and public bodies in the digital single market."* [2]. It is widely regarded as having a major impact on the world's corporations, with provisions for very large fines (up to the largest between 20 million Euros and 4% of a company's annual global revenue) and costs to comply that are estimated in the order of billions of dollars [146]. While the GDPR was introduced in the EU, its impact extends to the entire world as the legislation applies to the personal data of EU residents *"regardless of whether the processing takes place in the Union or not"* (Article 3(1); for more in-depth discussion on the GDPR's jurisdiction see [10]). It is worth noting that this legislation will be valid in the United Kingdom after its exit from the EU, as an equivalent legislation has been adopted in national law [1].

From a technical point of view, the GDPR applies to *"any information concerning an identified or identifiable natural information"*, including data *"which could be attributed to a person by the use of additional information"*, i.e., by taking into consideration methods that could be used to de-anonymize pseudonymous data (e.g., [288, 211]). Hence, the legislation applies when identification is *possible*, not only when it is actually performed. Cookies are expicitly mentioned: *"natural persons may be associated with online identifiers provided by their devices, applications, tools and protocols, such as internet protocol addresses, cookie identifiers or other identifiers [...]. This may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them."* Hence, the GDPR does not differentiate between first- and third-party cookies, and it is not hard to see how

the unique user identifiers that are present in many cookies—even session cookies, which only last for a single browser session—can be used to match personal information with the natural persons owning them.

Personal data should be kept for a short time: controllers have the duty of *"ensuring that the period for which the personal data are stored is limited to a strict minimum"*. Various practitioners [4, 78, 226] use a threshold of 12 months to ensure compliance with the law.

User Consent. Before the GDPR, websites were entitled to consider a simple visit to their pages as an implicit consent to accept any form of cookies. However, to comply with previous EU legislation [86], they often included small banners containing just an `OK` button to accept all cookies, or notes such as *"if you continue browsing, we will consider that you accept cookies."* Conversely, the GDPR clearly states that consent to tracking *"should be given by a clear affirmative act establishing a freely given, specific, informed and unambiguous indication of the data subject's agreement to the processing of personal data relating to him or her [...]. Silence, pre-ticked boxes or inactivity should not therefore constitute consent."* Moreover, websites are not allowed the option of refusing service to users who do not agree to tracking for purposes that are not essential to the functioning of the website: *"Consent should not be regarded as freely given if the data subject has no genuine or free choice or is unable to refuse or withdraw consent without detriment."*

Another important point is that consent should be given *prior* to data processing: websites cannot create non-strictly necessary cookies and then delete them if the user rejects them, because the cookies may have already been sent to the server if any request was performed after the cookie was created; for example, websites could implement this with a form of script blocking until the user explicitly consents the cookie creation.

Policies. The GDPR gives guidelines applicable to privacy and cookie policies. These should be transparent and expressed in a clear language: *"It should be transparent to natural persons that personal data concerning them are collected, used, consulted or otherwise processed and to what extent the personal data are or will be processed. The principle of transparency requires that information and communication relating to the processing of those personal data be easily accessible and easy to understand, and that clear and plain language be used."* Privacy policies, moreover, should be tailored to the specific website they apply to rather than being just taken from a template or from automated generators: *"the specific purposes for which personal data are processed should be explicit and legitimate and determined at the time of the collection of*

Table 6.1: Breakdown of websites analyzed by region and category.

	EU	USA	China	Others
Business	18%	52%	8%	21%
Entertainment	24%	53%	5%	18%
Finance	30%	27%	10%	33%
Food	27%	49%	10%	14%
Gaming	28%	53%	8%	11%
Government	28%	32%	–	40%
Health	18%	57%	16%	9%
Hobby	21%	56%	7%	16%
Kids	35%	57%	1%	7%
News	25%	35%	11%	29%
Politics	15%	65%	–	20%
Pornography	31%	52%	1%	16%
Real Estate	26%	37%	12%	25%
Religion	12%	63%	–	25%
Science	35%	47%	7%	11%
Shopping	35%	30%	10%	25%
Sports	48%	25%	1%	26%
Technology	11%	66%	9%	14%
Travel	37%	36%	9%	18%
Weapons	15%	72%	1%	12%

the personal data."

6.2 Methodology

The goal of our study is to verify to what extent the requirements discussed in §7.1 are currently implemented by websites. It is important to note that since the effects of the GDPR extend beyond the EU borders, we decided to perform a *global* measurement which can better capture the ability of users to opt-out from tracking on a worldwide scale. If the top-level domain (TLD) of a website corresponds to a country, we attribute that website to the country; otherwise, for international TLDs, we use IP geolocation to determine the country.

For each website we verify how much tracking is performed when users *never* consented to it, and rejected it when given the option. We also analyze

how difficult it is for users to opt out and what kind of interface and choices websites offer in this respect.

By design, our study cannot verify whether cookies are actually used to profile users; however, as discussed in §6.1.2, the GDPR applies as soon as cookies can *potentially* be used to identify users, even in the—arguably unlikely—case of websites that set cookie identifiers without using them later.

6.2.1 Domain Selection

We used the list of Alexa top-1M websites [252], divided in fine-grained categories through a proprietary solution offered by a security vendor. We then chose 20 of the most highly accessed categories; from each, we collected the top 100 entries in the Alexa list. All together, this resulted in a list of 2,000 websites: Table 6.1 on the facing page contains the list of categories and a breakdown by region. Of these 2,000 websites, 474 belong to the top 1K websites for worldwide traffic according to the Alexa ranking, 1,228 are ranked in the top 5K, and the remaining rank outside these boundaries.

6.2.2 Data Collection

Between July 6th and 30th 2018, we browsed the selected websites from EU IP addresses, recording which options they provided for their privacy and which cookies they created on the user's browser. We collected this information by filling a multi-step questionnaire that we implemented as a Chrome browser extension. The dataset was split evenly and each site was randomly assigned to one of the authors only after its exact interpretation–described in the following–was agreed and clarified in multiple in-person meetings. With our extension in place, we manually visited each website described above and, for each of them, recorded a number of features associated to the following five phases:

1 **Loading**. Before visiting each website, the extension deletes all cookies stored in the browser. Once the website is loaded, and before any interaction with it, the extension automatically saves the lists of newly created cookies and the list of fetched resources.

2 **Cookie Notice**. We manually classify the size and content of cookie notices. For size, we distinguish whether they are banners that allow navigation while they are present or "blocking" UI elements that preclude

browsing until they are dismissed. The categories for content are: a) `Anyway` for notices that just inform users that they will be tracked; b) `AutoAccept` for notices informing that, simply by visiting the website, users accept any cookies that it will set; c) `OnlyAccept` for notices having an accept button, but no quick way to reject tracking (a way to reject cookies is sometimes present through a link at the cookie settings dialogues); d) `AcceptReject` for notices that provide users with both an accept and a reject button; e) `JustSettings` for cases that lead users straight in a more complex settings dialog.

3 **Cookie Settings and Policy**. We manually browse to the cookie settings and privacy policy; we collect the full HTML of both and save it for later analysis. We also classify the kind of options available in the settings (e.g., to allow to reject by types of tracking purposes or to individually opt-out from each tracking company).

4 **Rejection**. We reject all tracking whenever possible, to detect if websites actually comply with user choice. In this step, we did not visit third-party services for opting out, which we analyzed as described in the following.

5 **Reload**. We reload the website to see the difference, if any, after the user explicitly rejected all tracking; our plugin saves again the list of cookies created and resources fetched. Finally, we check whether the website displays again a cookie notice and, if this is the case, we note its type as described above.

6.2.3 Third-Party Services

By analyzing cookie-related banners, settings, and policies, we repeatedly observed the presence of four popular third-party services used to opt-out from multiple tracking companies at once: *aboutads* [294], *youronlinechoices* [293], *networkadvertising* [210], and *TRUSTe* [274]. These services handle the whole process of opting in/out from third-party trackers and save the decision on cookies. We analyzed which companies are involved with each of them opting out from all the services and saving all the opt-out cookies created in the browser. In §6.3, we analyze how these services affect the real deployment of tracking cookies and how many websites link to them in order to opt out from tracking.

Table 6.2: Examples of cookie values and their "strength" as estimated by `zxcvbn`. We conservatively consider as identifiers only those with a strength whose base-10 logarithm is at least 9.

Cookie value	$\log_{10}$(strength)	Ent
`__test`	4.29	1.91
`12198692`	6.06	2.25
`W1TMYg**`	8.00	3.46
`6RqV3mB2nac`	**10.17**	3.45
`drZshFnT6g1M670owknOIA`	**22.00**	4.27
`2ee92b9f-2781-43a2-a326-af9cc922a942`	**35.67**	3.45

6.2.4 Recognizing Identifiers

The GDPR applies to pieces of data that can identify users, regardless of whether they are meant or used to actually track them. While many cookies are personal identifiers, not all of them are: for example, some record user preferences that are too coarse to identify users. While several cookies carrying little information can sometimes be combined to uniquely identify users, we take a conservative approach to avoid overestimating tracking.

A traditional approach to determine whether cookies carry enough information to identify users would be to measure their entropy—an approach that considers longer strings with several different characters as carrying more information. In our case, though, the entropy calculation may be misleading because it would attribute an excessive weight to common strings, such as dictionary words. Hence, we take inspiration from the literature on passwords [70, 69, 190, 285], which measures a password's strength—i.e., the amount of information it carries—as the number of guesses it would take an attacker to guess it. We use `zxcvbn` [285]—an approach that conservatively estimates a password's strength by evaluating the worst-case scenario of an attacker choosing the best approach from a set of possible ones—as our estimator; in Table 6.2 we show a few examples of cookie values and their strength output by `zcxvbn`. The Table also shows the entropy values even though, as expected, they do not provide a meaningful indication of the randomness of those cookies. We conservatively consider as identifiers cookies whose strength as a password is considered to be greater than 10^9 by `zxcvbn`, considering that they could be used to distinguish among at least 1 billion users. As timestamps are often stored in cookies, we ignore all parts of a cookie that contain a Unix timestamps

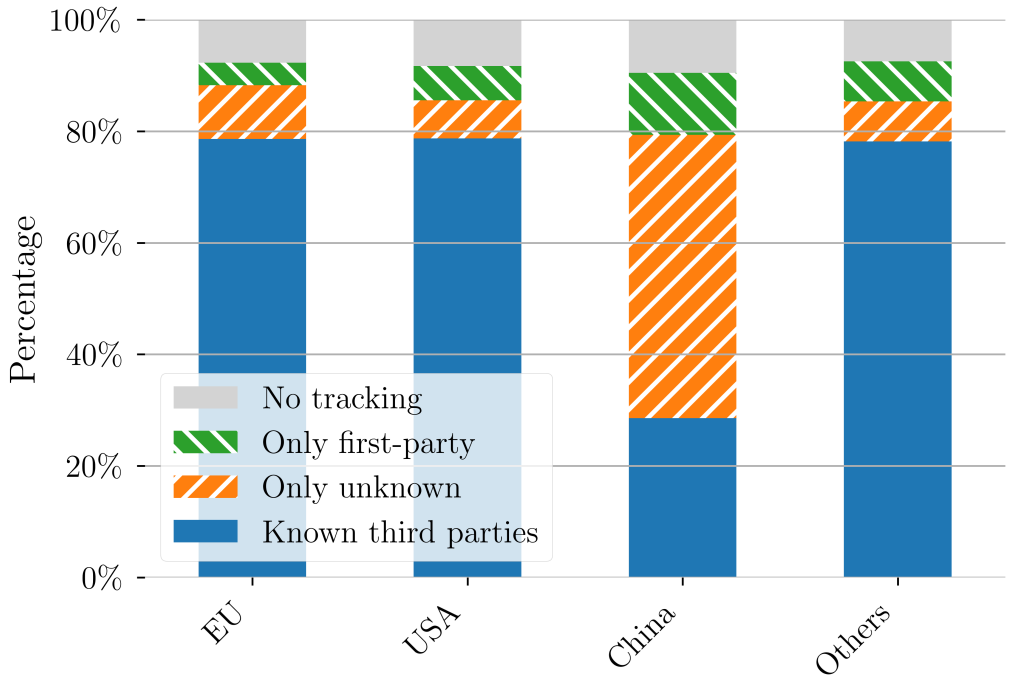


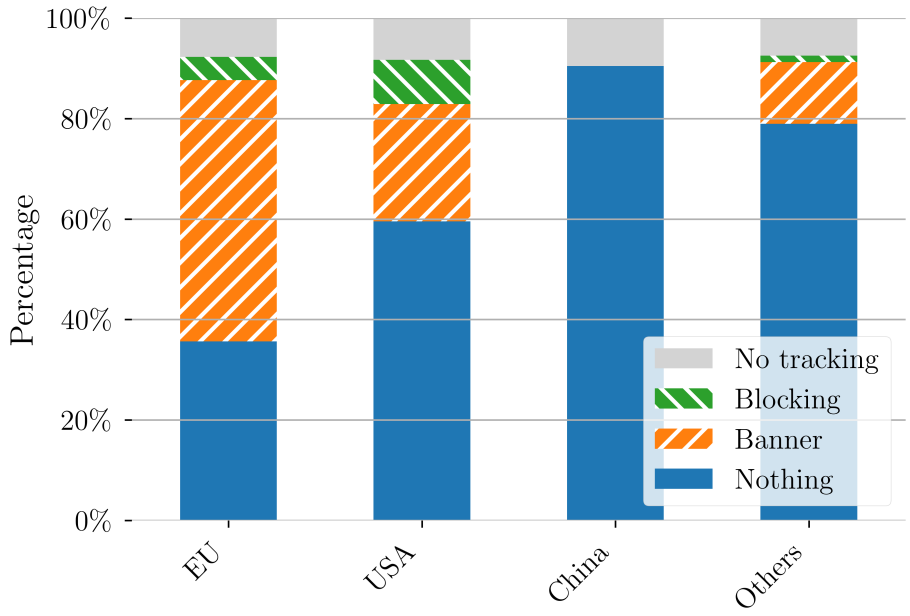
Figure 6.1: Kind of tracking observed.

in the period of our experiment. Again conservatively, we do not consider as identifiers cookie values that have been observed more than once in our whole dataset. After applying these rules, we found that 84% of the unique cookies we collect in our experiments were complex enough to identify a users, resulting in 54,803 unique cookies that we consider identifiers.

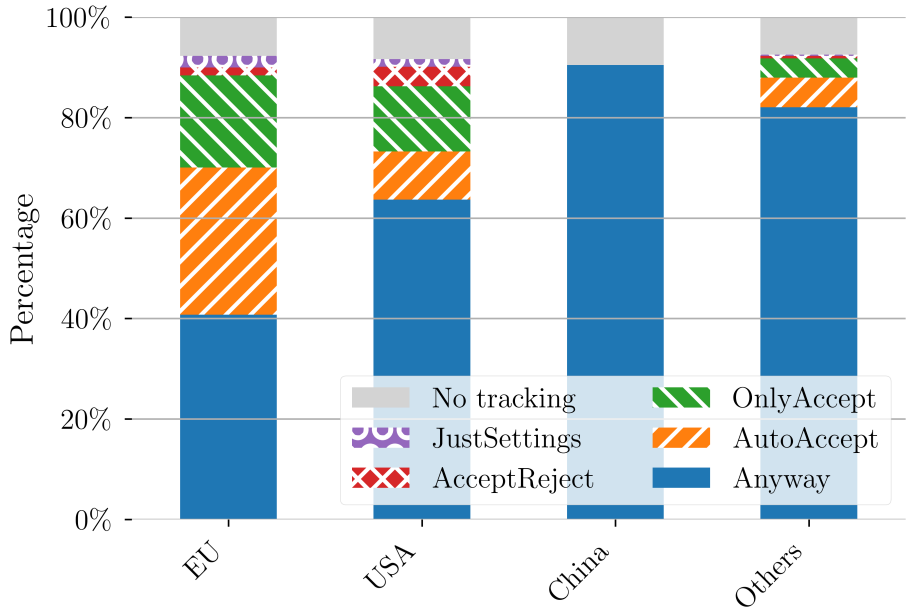
For each identifier cookie we also checked if its domain was a known third-party tracker (e.g., analytics, and advertisement), according to the list that Firefox uses for its tracking filter [105]. Our measurements in §6.3 show that this database is quite effective in categorizing third-parties in the USA and EU, but does not include many trackers we found in Chinese websites.

6.3 Data Analysis

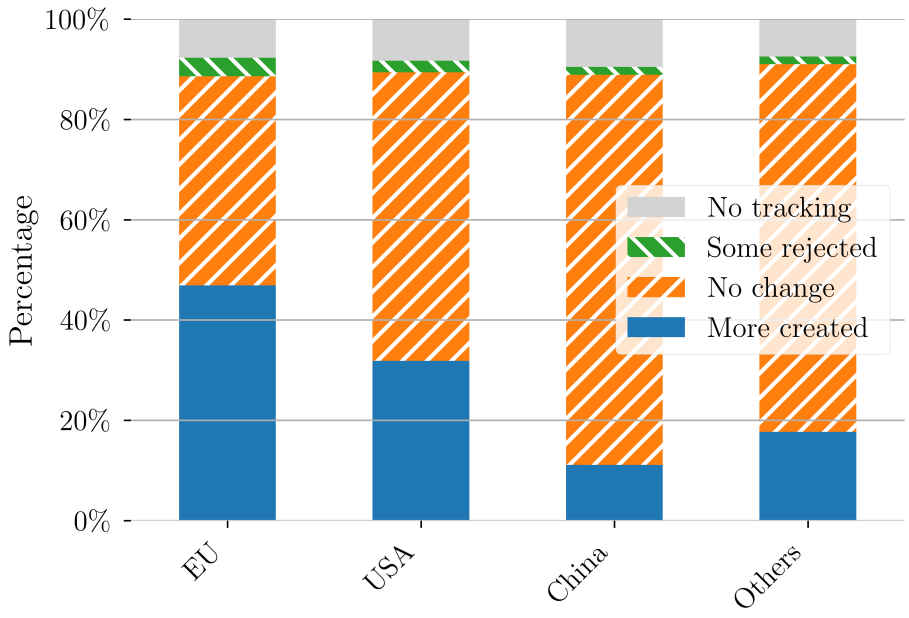
We present our evaluation results aggregated at different levels: we first discuss our general statistics using the whole data, we then breakdown the results by region, and finally by website categories.



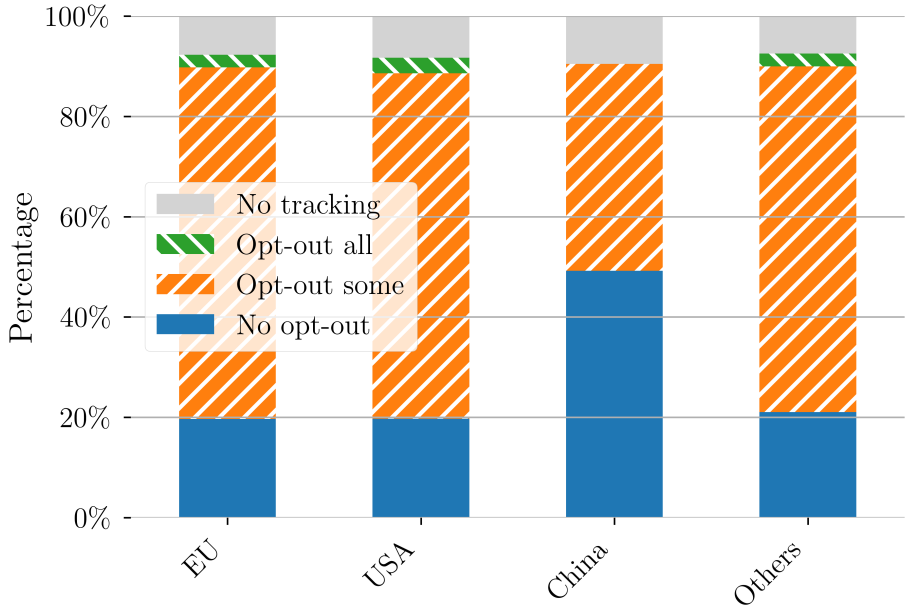
(a) Cookie notice size. While a relevant number of websites in EU and USA show cookie notices, almost no website hosted in China does.



(b) Type of cookie notice. Most websites leave no choice, inform that cookies will be set if navigation continues or give only the possibility to accept.



(c) Number of cookie identifiers after rejecting, if possible, and reloading.



(d) Number of cookie identifiers cookies after opting out from external websites.

Figure 6.2: Statistics about cookie user interfaces and tracking, broken down by region.

6.3.1 General Findings

We begin our discussion by highlighting a number of important points that apply to the entire dataset, independently from the geographic location or website category.

Most web pages perform tracking even if users do not give their consent. Figure 6.1 on page 108 shows that around 92% of the websites we considered perform some form of tracking—i.e., they set at least one identifier cookie (as discussed later, more than 80% of websites set cookies that last a year or more). This happens even before showing any banner about cookie policies, and even if the user chooses to opt-out from being tracked. Additionally, we observe that the vast majority of websites perform tracking using known third-parties services (generally for advertisements and analytics), and only 12% of domains rely uniquely on first-party tracking. Note that, as discussed in §7.1, the GDPR does not differentiate between first-party and third-party tracking; hence, users should have the right to reject both types of tracking to protect their privacy. We also see, as discussed in §6.2.4, that the tracker database we use does not include many third parties observed in Chinese websites (i.e., "only unknown" third-party domains).

Few websites provide an easy way to opt out from tracking. In Figure 6.2b on page 109 we break down the content of the cookie notice, if present. In some cases, users have no choice (i.e., the notice just mentions that cookies will be set); in others, they are informed that continuing navigation on the website will result in setting cookies (`AutoAccept`); some websites just have an "accept" button without a "reject" one (`OnlyAccept`). However, as we will see later, some of these cases may additionally include the option of settings. Unfortunately, the cases where users have a clear "reject" option (`AcceptReject`) or are presented right away with a cookie settings dialog (`JustSettings`) are, together, less than 4%.

Rejecting tracking is often ineffective. In Figure 6.2c we compare the number of cookies before and after the user rejected them and reloaded the website. In most cases, the number of cookies set by the server remains the same or even increases. The cases where some cookies are erased are, on average, 2.5%. The fact that *more* cookies can be created after rejecting them may appear counter-intuitive at first. However, this can be due to additional elements (e.g., ads) fetched when reloading the website, or to the fact that not the entire page content was loaded in the first place (e.g.,

Table 6.3: Statistics about tracking, cookie user interface and information, broken down by region.

Region Region	Tracking Tracking	Long-lasting ids	Third-party opt-out link	Browser Instructions	Cookie settings	Third-party notice
EU	92.3%	81.7%	32.0%	19.6%	17.7%	3.8%
USA	91.7%	80.1%	23.6%	6.6%	11.5%	4.7%
China	90.5%	82.5%	–	–	–	–
Others	92.6%	79.5%	8.5%	3.3%	3.3%	0.5%

because a blocking notice prevented to fully load the underlying webpage). In §6.5, we show the details of a case where more cookies are loaded after rejecting cookies.

Rejecting tracking is often ineffective. In Figure 6.2c we compare the number of cookies before and after the user rejected them and reloaded the website. In most cases, the number of cookies set by the server remains the same or even increases. The cases where some cookies are erased are, on average, 2.5%. The fact that *more* cookies can be created after rejecting them may appear counter-intuitive at first. However, this can be due to additional elements (e.g., ads) fetched when reloading the website, or to the fact that not the entire page content was loaded in the first place (e.g., because a blocking notice prevented to fully load the underlying webpage). In §6.5, we show the details of a case where more cookies are loaded after rejecting cookies.

Few websites have cookie settings. In Table 6.3, we observe that only 16% of EU websites and 12% of USA websites include a cookie settings interface. Numbers are even smaller for the other regions; for instance, none of the Chinese websites in our dataset had a cookie settings interface.

9 websites out of 10 create long-lasting identifiers. Even if the GDPR requires storing personal data for a minimum amount of time, it appears that this provision is rarely respected. In fact, Table 6.3 shows that around 90% of websites create identifiers lasting more than 12 month—the threshold we discussed in §7.1.

Third-party cookie notices are not very common. There are some third-party services that handle the cookie notice and, if present, the cookie settings. Therefore, we checked how many websites used services provided by the main third parties: OneTrust [220], TRUSTe [273], Quantcast [229], Cookiebot [4], and Evidon [87]. Table 6.3 shows that only 5% of USA websites use these services and the number is even lower in the European Union

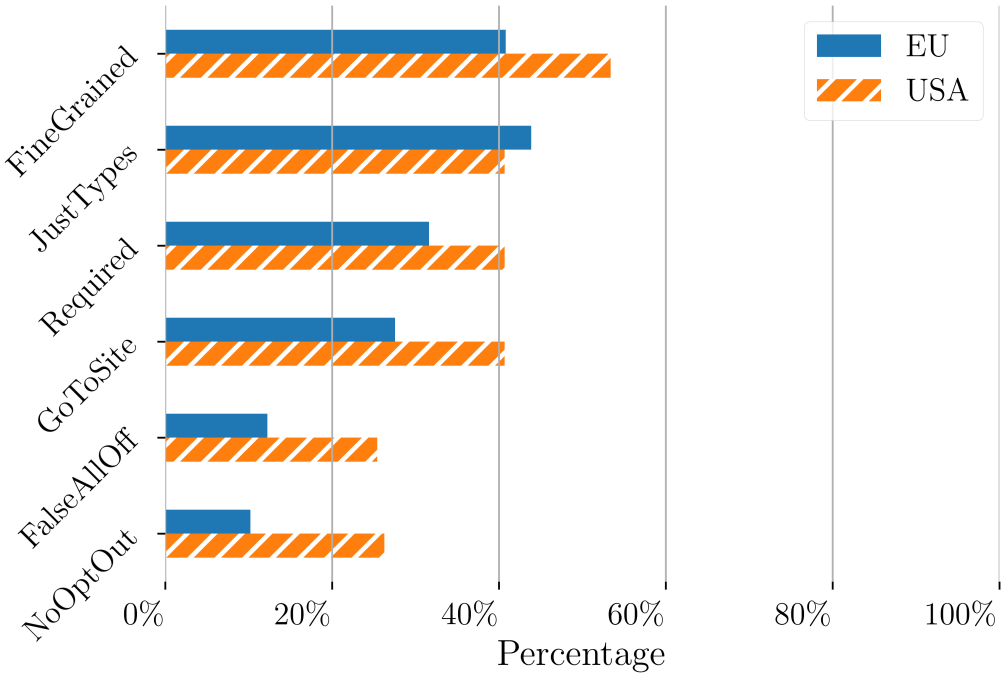


Figure 6.3: Characteristics of the settings dialogs.

(3%). These services are practically not used at all in the rest of the world.

Opting out through external services does not prevent all tracking. In Figure 6.2d we show that the number of identifying cookies varies when the user previously opts out through one of the third-party opt-out services included in §6.2.3. However, it is interesting to note that while the number of identifier cookies often decreases ("opt-out some"), the case in which all identifiers are blocked ("opt-out all") is lower than 3%.

The general picture we can draw from these results is that the overwhelming majority of the websites we visited track users even if, during our manual crawling, we never consented to cookies. This is not to say that GDPR had no effect: in fact, information about cookies is present in 33% of the websites, and it is often possible for the user to reject at least *some* of the tracking. However, from our tests it is hard to conclude if this is done on purpose or, as we will discuss in more details in §6.5, if this is a consequence of the challenge websites encounter when they want to prevent third-party tracking.

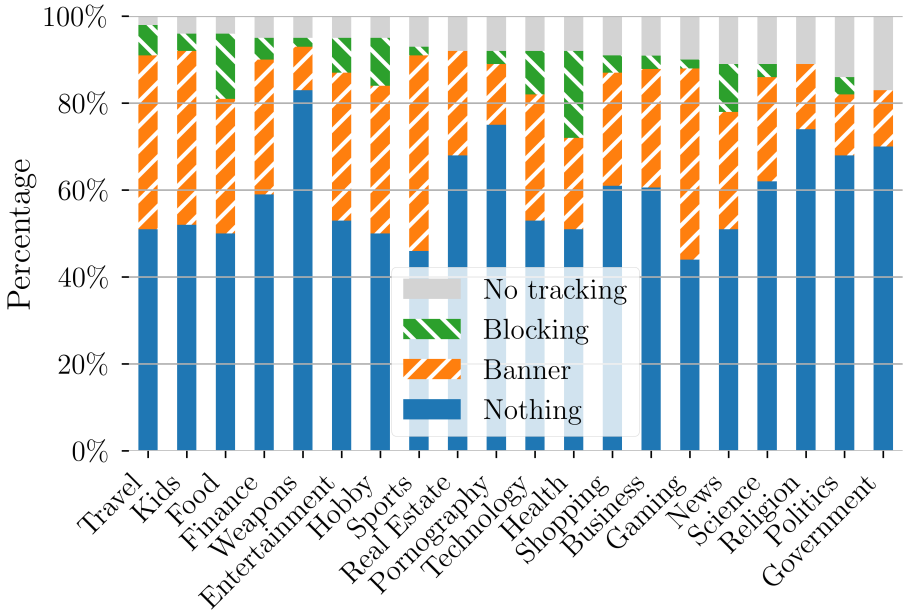
6.3.2 Regional Differences

We observe that the EU- and USA-based websites appear more compliant to the GDPR regulation; EU citizens that visit websites located in other parts of the world, on the other hand, may expect less in terms of privacy. Most of the websites we analyze are located in the European Union, USA, or China; we therefore group them in these three areas, plus one group referring to all other countries.

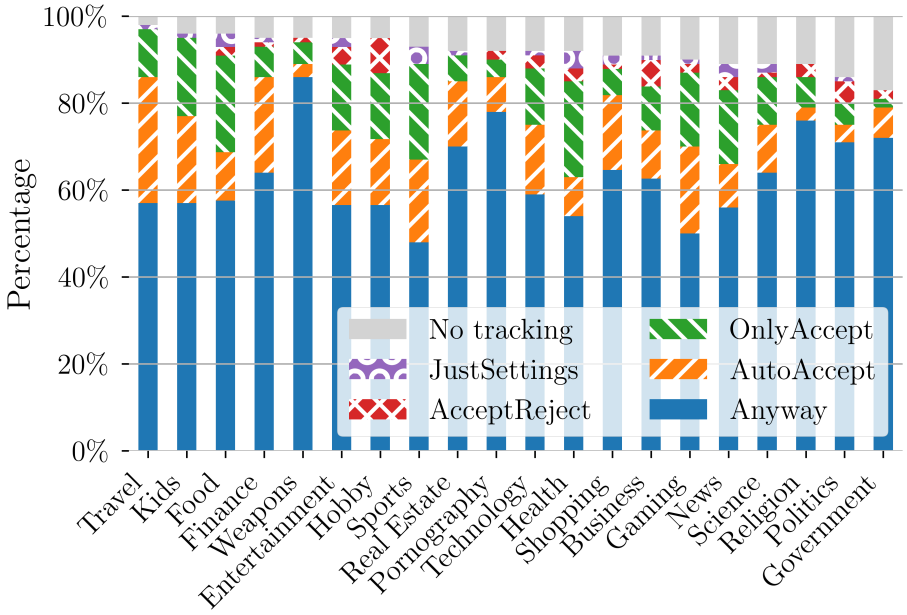
US-based websites look almost as impacted by the GDPR as the EU websites do. Throughout all of the information we present, it is apparent that websites in the USA appear to approach cookie regulations similarly to the EU. From Table 6.3 we see that the number of websites providing third-party opt-out links and cookie settings dialogs are similar between EU and USA; Figure 6.2a shows that cookie notices appear in 32% of US-based websites against the 57% of EU while the more intrusive "blocking" dialog is even more frequent in the USA than in the EU. Also Figure 6.2b shows that, even if cookie notices give more often control to the users in EU websites, the percentage of US websites that provide cookie control is still around 30%.

China-based websites appear not to care about the GDPR, but they are still impacted by it. From Table 6.3 and Figures 6.2a and 6.2b we see that Chinese websites do not seem to attempt complying with the GDPR, unlike many US and EU-based ones; yet, the result of Figure 6.2d is interesting because it proves that, if one opts-out from the global third-party services described in §6.2.3, they will be tracked less also when browsing 41% of Chinese websites.

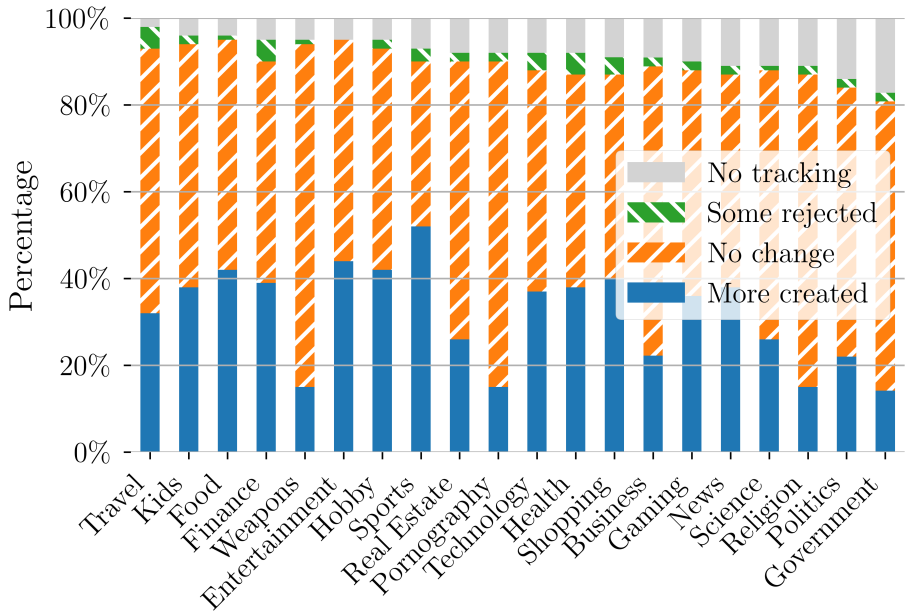
Cookie settings in the USA make opting out more difficult. In Figure 6.3 on the preceding page we show the types of and prevalence of cookie settings dialogs we observed. When users are allowed to opt-out from each of the ad providers or third-party trackers individually, we flag the cookies settings to be "FineGrained". In some cases ("JustTypes"), to simplify the user's job during the opt-out process, the settings dialog groups trackers by categories (e.g., advertisement, analytics, etc.). In other more desirable cases, the cookie settings interface gives both options. Often, there are so-called essential cookies that are required by the websites for functioning properly ("Required"). When opting out requires going to an external website, we set the "GoToSite" flag. Sometimes, there are shortcuts for opting out from "all" tracking, however, when we expand the list, we observe that



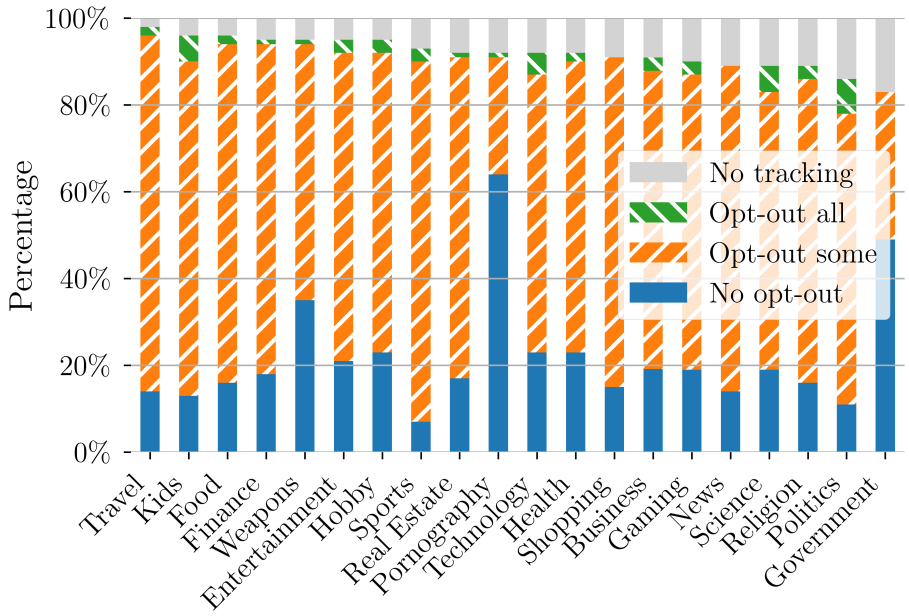
(a) Cookie notice size.



(b) Type of cookie notice.



(c) Cookie identifiers after rejecting tracking (if possible) and reloading.



(d) Cookie identifiers after opting out from external websites.

Figure 6.4: Statistics about cookie user interfaces and tracking, broken down by categories.

a significant number of third party services listed under the category does not allow the opt-out: we flag this as "FalseAllOff". For example, when browsing the details of the cookie settings, one sees that opt out must be performed by going to external websites, and cookies for which a way of opting out is just not provided ("NoOptOut"). Our results of Figure 6.3 show that cookie settings dialogs in the US are more complex, and that they generally require more work to opt out from all cookies.

Other Remarks. Cookie policies in the EU tend to give instructions on how to delete cookies from a user's browser more often than other websites ("browser instructions" in Table 6.3 corresponds to this kind of cookie settings dialogs); the results from the "other countries" aggregated category are, unsurprisingly, generally in the middle between the results of the USA and those of China. We do not breakdown these results into more fine grained regions because the number of websites would be too small to be statistically significant and representative. However, we observed that countries that are economically and/or geographically closer to EU-/USA/China have similar characteristics to the neighboring region.

6.3.3 Categories

We now compare how GPDR is adopted by different website categories. Figure 6.4 on the preceding page depicts details about type of cookie notice, cookie notice size, what happens after the user rejects tracking from the website and external websites for the 20 site categories of our analysis. Categories are sorted by the percentage performing tracking–i.e., those on the left have the highest percentage of websites that do tracking, while those on the right have the least. Most of the categories do not deviate much from the average behavior of all websites; we still observe that categories catering to more general audiences tend to give more information and control to the users (in particular, see Sports and Gaming in Figure 6.4a and 6.4b). In the following, we highlight some categories whose results significantly deviate from the mean; a breakdown of the kind of tracking we encounter on them is provided in Table 6.4 on the following page.

Pornography. Adult websites appear to behave very differently from others. While there is a considerable amount of websites in the category that perform some type of tracking, only less than 20% of them displays cookie notices (Figures 6.4a and 6.4b). Also, opting out from the services described in §6.2.3 reduces tracking only in less than 30% of the websites

Table 6.4: Kind of tracking for websites of particular categories.

Category	First-party	Unknown	Analytics	Ads	Content
All	76%	70%	70%	62%	41%
Pornography	69%	72%	**30%**	**39%**	**11%**
Weapons	79%	64%	62%	**45%**	38%
Religion	69%	50%	71%	58%	37%
Government	60%	**46%**	**41%**	**29%**	**17%**

(Figure 6.4d). We think this happens because this category has its own ecosystem and advertising networks [289], which appears to be quite segregated from the rest of the Internet. In this industry, the adoption of the GDPR rules appears to be progressing more slowly than on the rest of the Internet.

Weapons. This category has the most websites that perform tracking without informing users about it (over 80%). This might be due to the fact that these sites are mostly hosted in the USA, and—unlike other categories—appears to be directed mostly to a local audience, and not so much to EU citizens; this might explain why there is less interest in following the GDPR rules.

Religion. Over 70% of the websites in this category perform tracking but do not inform users. A possible interpretation of our results is that some of these websites may have been designed and implemented by volunteers rather than professionals; this could explain a slower adoption of the GDPR rules.

Government. Government pages are also largely not commercial, and this may explain why this is the category for which we observe the least tracking. Cookies could result from technical choices (e.g., including libraries, scripts or videos) rather than commercial ones; we hypothesize that in some cases website administrators might be unaware of some of the tracking done by their systems or of the relative consent requirements: this could explain the relative lack of information and/or control on cookie tracking.

6.4 Readability Analysis of Privacy Policies

Users often need to consult the cookie and privacy policies of the websites they visit in order to understand how their data is being processed, opt-out

Table 6.5: Readability scores and length of privacy policies per region. High FRES and low FKRL scores indicate easy texts.

Region	Policies	Avg. FRES	Avg. FKRL	Avg. Length (words)
USA	617	54.5	11.2	2,698
Europe	190	57.1	10.5	2,261
All	849	54.1	11.1	2,250

from tracking, or even to get information on how to change their browser configuration (e.g., disabling third party cookies). In fact, 59% of the websites in our study do not prompt their visitors with any privacy banner: in these cases, the available privacy options are only stated in the policy. Thus, it's critical that policies are readable and understandable, a requirement included in GDPR (see §7.1).

In this section, we measure the length and readability of privacy policies using two simple metrics, the Flesch Reading Ease Score (FRES) [98] and the Flesch-Kincaid Reading Level (FKRL) [152]. Both metrics estimate the text complexity using the average sentence length and number of syllables per word; they differ on the weighting factors they use and on their output. FRES emphasizes the word length and outputs a score up to 121.22, that is the easiest to understand possible text; it does not have a theoretical lower bound. For example, *Time* magazine has an FRES score around 52, and *Harvard Law Review* scores in the low 30s. On the other hand, the FKRL emphasizes on the sentence length, has a lower bound of -1.3 and no theoretical upper bound. The output of this test can be interpreted either as a US grade level or as the years of formal education required to understand a text (for scores higher than 10). Despite the simplicity of these formulas and the known caveats [139], both metrics have become a standard and are used by legislators and government agencies: for example, US legislators require insurance policies to have an FRES score above 45 [99]. Prior work used these tests to measure the complexity of privacy policies [143], and End User License Agreements (EULAs) [123].

We perform the readability analysis for the privacy policies of 849 websites written in English; 617 are from USA, 190 from the European Union and 42 from the rest of the world. Table 6.5 shows for each region, the average FRES, and FKRL scores as well as the average policy length. The distribution of the FRES scores averages at 54.1 with the standard deviation of 9.3 – minimum and maximum scores corresponding to 14,7 and

104.3. The easiest and hardest to read policies are both from websites in US: while the easiest belongs to a US government website, the two most difficult, which belongs to the same IT company, are from two online community websites. Documents for the general public should ideally score between 60 and 70 [123].

Prior work has calculated the FRES scores of 64 privacy policies [143], and reported an average score of 34.2. Similarly in 2007, Grossklag et. al calculated the average FRES score of 50 EULAs and reported a score of 35.7 [123]. Both works have used a dataset that is an order of magnitude smaller than ours and performed their measurements more than a decade ago. The FRES scores reveal an improvement on the readability of privacy policies over the years.

Our Flesch-Kincaid Reading Level exhibits a similar improvement. We obtain an average score of 11.1; therefore, a reader with at least 11 years of formal education should be able to understand how to opt-out from being tracked and the details about the cookies and privacy policy. Concerning that the average score of the privacy policies a decade ago was 14.2[143], the improvement over the previous set of policies is obvious. Despite the improvement on both of the metrics over the old privacy policies, the complexity of the privacy policies still remain high and outside the ideal range (FRES: 60–70; FKRL: 8–9) to be understandable by the general public. A comparison among regions shows that EU and US privacy policies have a similar complexity, with the European ones being slightly more readable.

As part of our policy readability assessment, we also measure the length of the privacy policies: on average, privacy policies consist of 2,250 words or 10 pages of double-spaced text. Considering the amount of websites a user may visit daily they are overly lengthy. Their size is very similar to the size of EULAs [123] that users tend to ignore [116]. Our findings regarding the size of the policies are very similar to what was reported in previous work.

Finally, we search the content of the privacy policies for any mention of the GDPR regulation. Over 84% does not make any mention of GDPR and 58% of those do not prompt the user with any type of privacy banner. This can be a possible indicator of websites that have not made any GDPR-related change. From the 16% (136) of the policies that mention the GDPR, 94 belong to US websites (11% of all policies from the US), 39 to EU (21% of the European ones), and 3 from the rest of the world (7%). From the 3 policies outside US and Europe, 2 belong to media conglomerates in India, and one to an Australian bank.

6.5 Case Studies

In this section, we will first (§6.5.1) focus on a couple cases, to highlight some particular and perhaps surprising results for tracking that we observed in our evaluation, then (§6.5.2) we discuss some third-party opt-out services. Finally, (§6.5.3) we discuss some example of user interfaces, highlighting those that we consider good, bad, and misleading.

6.5.1 False Rejections

As shown in §6.3, most websites do not let visitors avoid tracking. In fact, more than 90% create tracking cookies immediately after users load the website, even before they are able to take any decision with respect to tracking. In the following, we present the cases of two large companies whose websites implemented cookie control incorrectly, either on purpose or in consequence of a bug in programming.

Rejecting cookies does not impact tracking. The website of a major company in the food and beverages business has a banner on the bottom of the home page stating *"[COMPANY NAME], 'we', envisages to use certain categories of cookies for several reasons, but need your agreement to do so"* and *"We will not use any categories of cookies other than those that are strictly necessary for the website to function if you do not elect to opt-in to those categories of cookies"*. While a visitor would think that this website would allow the user to opt-out from tracking, this is not the case. In reality, this website creates all the tracking cookies before any consent is given, and the answer to the banner (e.g., accept or reject) does not have any effect on the already present cookies. No matter whether one rejects or gives consent, the website just creates a new cookie with the name/key `OptanonAlertBoxClosed` and the current date as value. In addition to this new cookie, the website continues to keep the tracking cookies from companies such as Gigya, Google or Adobe. For instance, the website includes the `ucid` cookie from Gigya, described by them as a *"Unique computer identifier used for generating reports, and used by the Web SDK to get saved response"*. This cookie has a time-to-live of over a year.

More cookies created after rejecting tracking. A major website directed to software developers includes a blocking banner that only allows to browse the website once the user accepts or rejects tracking cookies. The banner indicates *"Click 'I Accept' below to consent to the use of this technology across*

the web", after explaining that they use cookies for advertisement and analytics. Before any decision is taken, the website startlingly creates a cookie named `VISITOR` with a unique user identifier. However, things do not end here: if the user rejects cookies, the `VISITOR` cookie will not be deleted, but multiple new tracking cookies from third-party advertisers are created—one of them described in their website as *"one of the main advertising cookies"* of the company. In fact, the website implemented a script blocker: some cookies were never created before any choice was made, but not all the tracking scripts were suspended.

6.5.2 Third-party Opt out

In §6.3, we discussed four third-party opt-out services (aboutads, youronlinechoices, networkadvertising, and TRUSTe) designed to help websites and their clients opt-out from third-party tracking when desired. We tested them for three days in a row and measured the time it took to reject cookies, and the number of companies from which we could successfully opt out from the list they offered.

We observed that these websites were rather slow, making external requests that require between 2 and 5 minutes for each service–i.e., opting out from all the four requires 8–20 minutes. Each of these services claims to let the user opt out from a list of 90–244 third party trackers; however, several of them returned errors, and on average we were only able to opt out from 85% of the "supported" services.

6.5.3 User Interfaces: the Good, the Bad and the Ugly

In this Section we present a detailed, manual analysis of three popular websites and their implementation of the GDPR user consent notice.

The Good. When accessing an important technology website, a banner that blocks the interaction with the website until a choice is made appears. This notice includes three options: "I ACCEPT", "I DO NOT ACCEPT" and "More Options". The user can choose whether to accept or reject all cookies with just one click, which we consider definitely useful and straightforward. There is also an option letting users define which specific cookies they accepts and which they do not. The option pane further provides the user with two different groups: first-party and third-party cookies. Each of these groups is further broken down in five sub-groups: "Information storage and access", "Personalisation", "Ad selection, delivery, reporting",

"Content selection, delivery, reporting", and "Measurement". Finally, the user can even "See full vendor list" and opt out from each of them independently. In our opinion, this is a clear example of how websites can and should give users full control over the tracking cookies created on their browsers.

The Bad. A large news website does not show any banner to the user indicating cookie usage related to tracking. There is a clickable text at the bottom of the website called "Privacy"—reachable only after scrolling through more than 25 pages of content. After finding the link and following it, there is no quick way to opt out from the tracking cookies, as the website just informs the reader about a third-party opt-out service [293]. This is a clear example in which users can stop tracking cookies somehow, by they do not have a simple way of doing it.

The Ugly. An important gaming website provides a user interaction that we found reminiscent of "dark patterns", i.e., "*tricks used in websites and apps that make you buy or sign up for things that you didn't mean to*" [40]. This website includes a blocking notice with two options: a big button with "I ACCEPT" and a smaller "Show Purposes" link. When clicking the second option, a list of categories appears; if the user marks some or all of them and clicks "Save and continue", a big "Accept all" button appears followed by long list of companies. If the user avoids the button, scrolls down to the end of the list, and hits the small "Reject all" link, an "Are you sure?" dialog appears, warning the user that rejecting tracking could result in a "limited experience". A "Go back" button brings back to the previous windows where users can reconsider their opinions about tracking; the only way to actually enter the website without giving consent to tracking is instead clicking the smaller "Leave" link. If any of the accept buttons is ever pressed during the process, instead, the user is directed straight away to the website without any additional formalities.

6.6 Discussion

Our results show that many websites, even outside the EU, attempt to comply to the GDPR regulations. Even websites that did not (such as the majority of Chinese websites) are affected by its impact, as opting out from third-party services can also reduce the amount of tracking on those websites.

Nevertheless, compliance is only superficial and we still found that a

large majority of websites track users through cookies, and that the spirit of the law is often still not applied. This can be the consequence of legal, economic, and technical reasons. With respect to legislation, it currently deals with personal data in general, without much detail about how it should be interpreted in particular use cases (it is worth mentioning that the word "cookies" appears exactly once in the 88-page document of the GDPR). Moreover, specific technical guidelines are still missing. From an economic point of view, stakes are big: on the one hand, fines imposed because of non-compliance to the GDPR can be very large; on the other hand, though, the main source of income of many websites is advertising, and we speculate that letting users easily opt out from tracking could negate them a part of their income that could potentially be even larger than the fines they could face. Uncertainty with respect to the jurisdiction and enforcement of the regulation are also involved. Finally, from a technical point of view, enforcing a user-specified opt-out can be very difficult, in particular when a website includes external content (say, a video or a script). In fact, the current HTML specification does not let a website specify that websites providing third party resources should not track users, according to the preferences they expressed. The result is that some websites we visited were forced to provide only a static, text-only page to users who opted out from tracking.

Our conclusion is that, as of now, web cookie tracking is still far from what privacy advocates and drafters of the law envision.

6.7 Related Work

Tracking. Web tracking has been the focus of several pieces of work. Krishnamurthy et al. [158] were among the first to analyze tracking related to HTML cookies in 2009. One year later, Eckersley [79] realized that users could be identified using certain information from the browser, such as the `user-agent` or the size of the screen.

In the following years, new techniques appeared to fingerprint the user, such asusing WebGL rendering to obtain a precise identifier [48] or analyzing the extensions installed on the browser [244]. Recent studies analyzed how good fingerprinting is in the current Internet [162, 278], showing that some techniques are more useful and stable than others.

Many other works analyzed the prevalence of both fingerprinting and cookie tracking. Results show that fingerprinting is not as widespread as

general cookie tracking, but is still commonly used by many tracking companies [215, 14]. Recent work by Englehardt et al. [83] and Sanchez-Rola and Santos [243] show that most websites have some kind of tracking either via cookies or fingerprinting, indicating a growing popularity for these techniques.

Two very recent pieces of work evaluate the changes in the tracking panorama induced by the GDPR. Degeling et al. [68] performed a longitudinal study comparing the information presented to users of EU websites before and after GDPR, focusing on the changes in privacy policies and information presented to users. We complement these results by observing websites *outside* the EU and by studying the tracking that websites actually perform, and the chasm with what they communicate to users. The *whotracks.me* database [150], based on results from real users running a privacy-oriented browser or browser plugin, has been updated with results remarking that the GDPR induced consolidation in the advertising market, increasing the market share of the biggest players while reducing that of smaller competition [286]. Our results differ from these latter because: 1 that dataset is based on a list of known third-party trackers [57], which by design does not include first-party tracking and may miss some third parties as we discuss in §6.2.4; 2 we evaluate tracking that happens when users *do not* consent to it, while information on user consent is absent from the *whotracks.me* data.

Policies. Prior work has tried to increase privacy policy transparency by annotating and extracting the most essential parts of privacy policies. Zimmeck S. and Bellovin S. [297] proposed Privee, an architecture for automatically extracting essential policy terms and present them to a user using a combination of crowdsourcing and machine learning classification techniques. Similarly, Oltramari et al. [217] proposed a framework for annotating policies using a combination of crowdsourcing, machine learning and natural language processing. Other works, measured the complexity of privacy policies and their perception by end users. McDonald et al. [188] compared different privacy policy formats and evaluated the understanding of privacy policies by 749 Internet users. They report that users were unable to reliably understand companies' privacy policies and all format were simlarly disliked by users. Jensen C. and Pots C. [143] analyzed the content and the complexity of 64 privacy policies from popular websites. Their measurements shows that privacy policies were more complex a decade ago than now. Grossklags J. and Good N. [123] evaluated

the readability and usability of 50 EULAs from popular programs. Their results shows that complexity of EULAs is comparable to privacy policies. Also, their user study shows that notice and consent by the user during software installation is not sufficiently achieved.

6.8 Conclusion

We presented an evaluation of the the practice of tracking users on web after the new European General Data Protection Regulation (GDPR) entered in effect. Our goal was not to judge compliance to the law, but rather investigating to what extent the introduction of this law helps users better control their privacy.

We found that this regulation has a global reach, in particular impacting US-based websites similarly to those in the European Union; in addition, third party opt-out services also affect tracking on websites that did not explicitly attempt to comply with the new law. On the other hand, we find that despite this, tracking is still ubiquitous and present in more than 90% of websites, even those in the EU. We think that the panorama of tracking in the Web is in flux, and the future results will depend on the technical and standards-based solutions that will make it easier to block tracking, on the enforcement of current regulation, and on the new laws that will be drafted in Europe and in the rest of the world.

Part III

Offensive Web Security and Privacy Techniques

"I have to try and change the landscape, whatever it is."

Robert Plant (1948–)

Chapter 7

Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies

Browser extensions are the most popular technique currently available to extend the functionalities of web browsers. Extensions exist for most of the browser families, including major web browsers such as Firefox, Chrome, Safari, and Opera. They can be easily downloaded and installed by users from a central repository (such as the Chrome Web Store [117] or the Firefox Add Ons [200]).

Unfortunately, extensions are also prone to misuse. In fact, due to their close relationship to the browser environment, they can be abused by an adversary in order to gather a wide range of private information — such as cookies, browsing history, system-level data, or even user passwords [43]. Due to this raising concern, the amount of research studying the security implications and vulnerabilities of browser extensions has rapidly increased in the last years [72, 31, 125, 29, 49, 174, 148].

When browser extensions were first introduced, websites were able to access all their local resources. As a consequence, malicious actors started

to use that freely-accessible data to enumerate the extensions a user has installed in her system, or even to exploit vulnerabilities within installed extensions [156]. To mitigate this increasing threat, Firefox introduced the `contentaccessible` flag and Chrome a new manifest version [118] to implement some form of access control over the extension resources. In the rest of the chapter we will refer to these security measures as *access control settings*. Developers of Safari decided to adopt a different mechanism, which consists in randomizing at runtime part of the extension URI [21]. We will refer to this second class of protection technique as *URI randomization*.

Information of the web browser has been used for a number of malicious or "questionable" purposes. For example, *Panopticlick* [79] creates a unique browser fingerprint using the installed fonts, among other features. *PluginDetect* [113] retrieves instead the list of plugins installed in the browser. Even worse, this technique has recently been used in two reported *fingerprinting-driven malware* campaigns [249, 271].

Thanks to the existing browser security countermeasures described, so far extensions were protected against these fingerprinting techniques. Two very simple enumeration attacks were recently proposed to retrieve a small number of installed extensions in the browsers that adopted access control settings [145, 42]. These techniques took advantage of accessible resources of the extensions present in Chrome and Firefox to identify a small number of popular extensions. In addition, XHOUND [260] was also recently proposed to enumerate extensions and perform fingerprinting, by measuring the changes in the DOM of the website.

In this chapter we present the first in-depth security study of all the extensions resource control policies used by modern browsers. Our analysis show that **all browsers families** that currently support extensions are vulnerable to some form of enumeration attack. In particular, while the two design choices (i.e., access control settings or URI randomization) are both secure from a theoretical point of view, their practical implementation suffers from many different problems.

We discuss two offensive techniques to subvert these control policies, one based on a timing side-channel attack and one based on an involuntary leakage of the random URI token that affects many extensions. At the time of writing, these attacks undermine the extension security of all browsers. We also discuss a set of attacks based on these techniques, which allow third-parties to perform precise user fingerprinting, or to perform vari-

ous types of targeted attacks, performing proof-of-concept tests of some of them.

We already reported the discovered problems to the involved browsers and extensions developers and we are currently discussing with them about possible fixes.

The remainder of this chapter is organized as follows. §7.1 provides the background on extension control methods. §7.2 describes the problems and two different attacks to subvert them. §7.3 describes the impact of the problems in a broad set of scenarios. We then discuss possible countermeasures and summarize the outcome of our research in §7.4. Finally, §7.5 discusses related work and §9.7 concludes the chapter.

7.1 Background

All browsers that support extensions implement some form of protection to prevent arbitrary websites from enumerating the installed extensions and freely accessing their resources. After an extensive survey of several traditional and mobile browser families, we identified two main classes of protection mechanisms currently in use: *access control settings* (§7.1.1), and *URI randomization* (§7.1.2).

7.1.1 Access Control Settings

The most popular approach to protect extension resources from unauthorized accesses consists in letting the extensions themselves specify which resources they need to be kept private and which can be made publicly available. All browsers that adopt this solution rely on a set of configuration options included in a manifest file that is shipped with each extension. For security reasons, by default all the resources are considered private. However, developers can specify in the manifest a list of accessible resources.

This solution is currently used by all browsers based on Chromium, all the ones based on Firefox and Microsoft Edge.

Chromium family

The Chromium family includes all versions of Chromium (such as Google Chrome), and all browsers based on the Chromium engine (e.g., Opera,

```
"name": "description",
"example": "Example extension",
"version": "1.0",

"browser_action": {
  "default_icon": "icon.png",
  "default_popup": "popup.html"},

"permissions": [
  "activeTab",
  "https://ajax.googleapis.com/"],

"web_accessible_resources": [
  "images/*.png",
  "style/double-rainbow.css",
  "script/double-rainbow.js",
  "script/main.js",
  "templates/*"], ...
```

Figure 7.1: Snippet of a Chrome Extension `manifest.json` file.

Comodo Dragon, and the Yandex browser).

Extensions in this family are written using a combination of HTML, CSS, and JavaScript [119]. They are not required to use any form of native code, as it is instead the case for plugins or other forms of browser extensions. Each Chromium extension includes a JSON file called `manifest.json` that defines a set of properties such as the extension name, description, and version number (see Figure 7.1 for an example of manifest). The manifest is used by the browser to know the functionality offered by the extension and the permissions required to perform those actions [118].

In the first version of the manifest, there was no restriction over the resources of the extensions accessible from third-party websites. Because of that, different tools were released to take advantage of this weakness to enumerate user extensions and exploit their vulnerabilities [156]. To mitigate this threat, Google decided to introduce dedicated access control settings in the second version of the manifest file. This extension uses a parameter (`web_accessible_resources`) to specify the paths of packaged resources that can be used in the context of a website. Resources are available through the URL `chrome-extension://[extID]/[path]`. However, any navigation access to an extension or its resources is blocked by the browser, unless the extension resource has been previously listed as accessible in its man-

ifest.json. This solution was explicitly designed to minimize the attack surface while protecting users' privacy.

Firefox family

Firefox family extensions (or Add-ons, as they are called in the Mozilla jargon) can add new functions to the web browser, change its behavior, extend the GUI, or interact with the content of websites. Add-ons have access to a powerful API called XPCOM [205], that enables the use of several built-in services and applications through the XPConnect interface. In the Firefox family (which includes for example Firefox Mobile, Iceweasel and Pale Moon), extensions are written in a combination of JavaScript and XML User Interface Language (XUL). Extensions are also allowed to use functionality from third-party binaries or create their own binary components. Recently, Mozilla changed its extension development framework, introducing the Add-on SDK of the JetPack project [202]. This development kit provides a high-level API, easing the development process and addressing some of the security issues of previous Firefox extensions.

The registration and allocation of the different extensions is performed through the *Chrome Registry* [201] which is also in charge of customizing user interface elements of the application window that are not in the windows content area (such as toolbars, menu bars, progress bars, or windows title bars). Each extension contains a chrome.manifest file that specifies options related to three main categories — content, locale, and skin — as exemplified in the following snippet:

```
content  ext                 src/content/
skin     ext  classic        src/skin/
locale   ext  en-US          src/locale/en-US/
content  pck  chrome/ext/pck contentaccessible=yes
```

As it was the case for Chromium extensions, originally there was no control performed to prevent external websites from accessing the different resources of an extension. And also in this case, developers decided to solve the problem by including a new option in the chrome.manifest (called contentaccessible and depicted in the last line of the previous example) that specifies which resources can be publicly shared. However, resources have a restricted access by default, unless contentaccessible=yes is specified in the manifest.

```
"applications": {
  "gecko": {
    "id": "{the-addon-id}",
    "strict_min_version": "40.0.0",
    "strict_max_version": "50.*"
    "update_url": "https://foo/bar"
  }
} ...
```

Figure 7.2: Snippet of a Firefox WebExtension manifest's new data.

Firefox is now developing a new way of handling Add-ons with the name of WebExtensions[204]. This technology is designed mainly for cross-browser compatibility, supporting the extension API of Chromium. Porting extensions between the two platforms will require few changes in the code of the Add-on. The new extensions will also use a `manifest.json`, including some extra data specific for Firefox (see Figure 7.2). In order to access the different resources of the extension, Firefox will use the `moz-extension://` schema.

As WebExtensions are currently in an early stage we are not including them in our tests, but we notified their developers and we will discuss more about them in §7.4.

Microsoft Edge

Edge will be the first Microsoft browser to fully support extensions. It will follow a Chrome-compatible extension model based on HTML, JavaScript and CSS. This means that the migration to Microsoft Edge for Chrome extension developers will require minimal effort.

Beside the general web APIs, a special extension API will provide a deeper integration with the browser, making possible to access features such as tab and window manipulation. The manifest will be named `manifest.json` and will use the same JSON-formatted structure and general properties of the Chromium implementation. The URL to access the extension resources follows the `ms-browser-extension://[extID]/[path]` schema.

As the design is in its preliminary stages and it is not yet fully working, we are not including it in our analysis.

```
1 <script type = "text/javascript">
2 var myImage = safari.extension.baseURI +
3 "Images/paper.jpg";
4 document.body.style.cssText =
5 "background-image: url(" +myImage+ ")";
6 </script>
```

Figure 7.3: Example of background image load in CSS using absolute URLs in Safari extension.

7.1.2 URI Randomization

As Safari was one of the last major browsers to adopt extensions, its developers implemented a resource control from the beginning to avoid enumeration or vulnerability exploitations of installed extensions. Instead of relying on settings included in a manifest file like all the other major browsers, Apple developers adopted a URI randomization approach. In this solution there is no distinction between private or public resources, but instead the base URI of the extension is randomly re-generated in each session.

Safari extensions are coded using a combination of HTML, CSS, and JavaScript. To interact with the browser and the page content, a JavaScript API is provided and each extension runs within its own "sandbox" [20]. To develop an extension, a developer has to provide: (i) the global HTML page code, (ii) the content (HTML, CSS, JavaScript media), (iii) the menu items (label and images), (iv) the code of the injected scripts, (v) the stylesheets, and (vi) the required icons.

These components are grouped into two categories: the first including the global page and the menu items, and the second including the content, and the injected scripts and stylesheets. This second group cannot access any resource within the extension folder using relative URLs as the first group does. Instead, these extension components are required to use JavaScript to access the randomized URI that changes each time Safari is launched. Absolute URIs are stored in the `safari.extension.baseURI` field, as shown in Figure 7.3.

7.2 Security Analysis

In the previous section we presented the two complementary approaches adopted by all major browser families to protect the access to extension re-

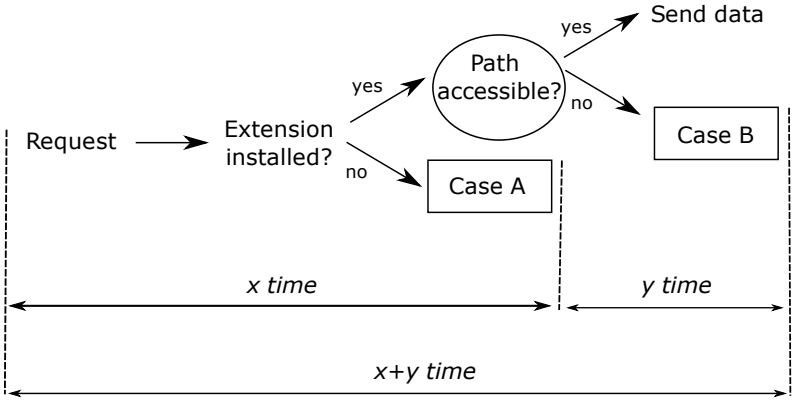


Figure 7.4: Resource accessibility control schema.

sources. The first solution relies on a public resource URI, whose access is protected by a centralized code in the browser according to settings specified by the extension developers in a manifest file. The second solution replaces the centralized check by randomizing the base URI at every execution. In this case, the extension needs to access its own resources by using a dedicated Javascript API.

While their design is completely different, both solutions provide the same security guarantees, preventing an attacker from enumerating the installed extensions and accessing their resources. We now examine those two approaches in more detail and discuss two severe limitations that often undermine their security. It is important to note that these attacks can also be used in any type of device with a browser with extension capability, such as smartphones or smartTVs.

7.2.1 Timing Side-Channel on Access Control Settings Validation

As already mentioned, the vast majority of browsers adopt a centralized method to prevent third parties from accessing any resource of the extensions that have not been explicitly marked as public. Therefore, when a website tries to load a resource not present in the list of accessible resources, the browser will block the request. Despite the fact that, from a design point of view, this solution may seem secure, we discovered that all their implementations suffer from a serious problem that derives from the fact that these browsers are required to perform two different checks: (i) to verify if a certain extension is installed and (ii) to access their control set-

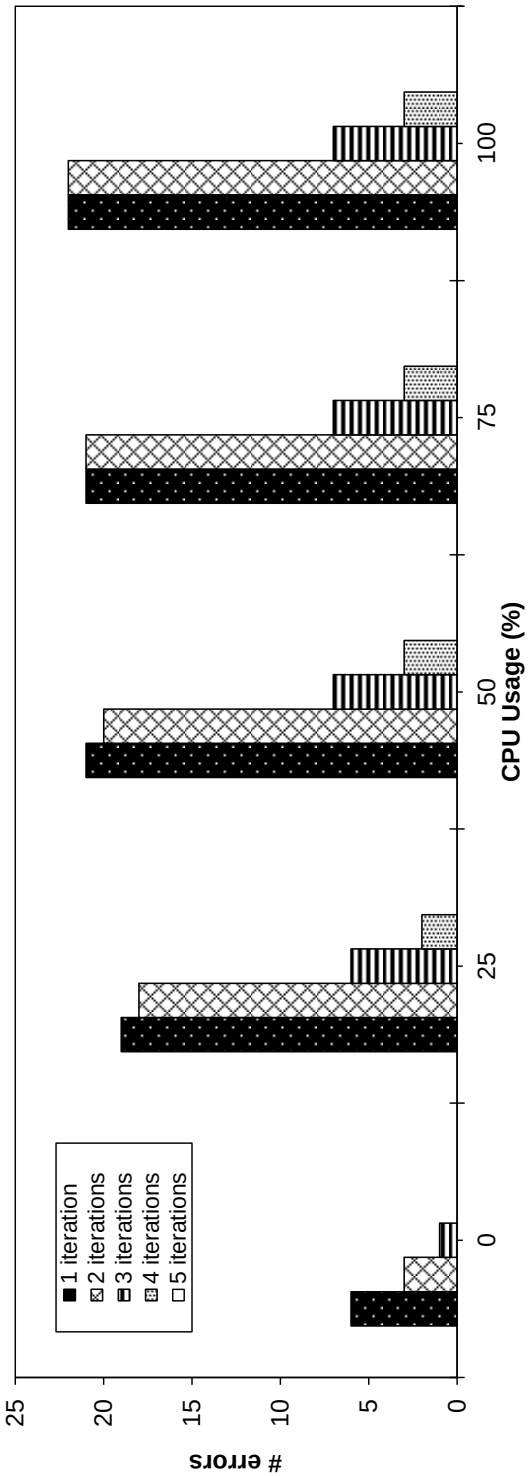


Figure 7.5: Comparison between number of iterations and errors with different CPU usages (%), .

tings to identify if the requested resource is publicly available (see Figure 7.4 for a simple logic workflow for the enforcement process). When this two-step validation is not properly implemented, it is prone to a timing side-channel attack that an adversary can use to identify the actual reasons behind a request denial: the extension is not present or its resources are kept private. To this end, we used the *User Timing API*[1], implemented in every major browser, in order to measure the performance of web applications.

As an example, an attacker can code few lines of Javascript to measure the response time when invoking a fake extension (refer to case A in Figure 7.4). For instance, in Chromium the requested URI could look like this:

```
chrome-extension://[fakeExtID]/[fakePath]
```

Then, the attacker can generate a second request to measure the response time when requesting an extension that actually exists, but using a non-existent resource path (case B in Figure 7.4):

```
chrome-extension://[realExtID]/[fakePath]
```

By comparing the two timestamps, the attacker can easily determine whether an extension is installed or not in the browser. Similar response times mean that the central validation code followed the same execution path on the two requests and, therefore, the extension is not installed in the browser. Otherwise, significantly different execution times mean that only the second test failed and, therefore, that the requested extension is present in the browser.

We performed an experiment in order to empirically tune the time difference threshold and the number of correct requests required to ensure the correctness of our attack. In particular, the following configuration was used:

- We configured 5 different CPU usages: 0%, 25%, 50%, 75%, and 100%. The experiment was executed on a 2.4GHz Intel Core 2 Duo with 4 GB RAM commodity computer.
- The attack was configured to be repeated from 1 to 10 iterations. Note that each iteration performs two calls to the browser: one that asks for the fake extension and one that asks for the actual extension with a fake path.

[1] https://www.w3.org/TR/user-timing/

- We repeated each attack testing 500 times to avoid any bias. In this way, we performed: 2 calls × 10 iteration configurations × 500 times × 5 CPU usages, resulting in a total number of 275,000 calls.

We observed that, when the execution paths were different, the response times differed by more than 5%. It is important to remark that our method exploits the proportional timing difference between calls rather than using a pre-computed time for a specific device. Figure 7.5 shows the precision across different CPU loads and different numbers of iterations. Five iterations were sufficient enough to achieve a 100% success rate even under a 100% CPU usage.

Affected Browsers

We tested our timing side-channel attack on the two families (Chromium-based and Firefox-based) that use extensions access control settings.

Our experiments confirm that all versions of Chromium are affected by this vulnerability. Browsers such as Chrome, Opera, the browser of Yandex (largest search engine in Russia) and the browser of Comodo (largest issuer of SSL certificates) are included in this group. As aforementioned, we are not including Edge and Firefox WebExtensions because they are still in early stages of development. However, as they follow the same extension control mechanism as Chromium, they are also likely to be vulnerable to our timing side-channel attack.

Surprisingly, non-WebExtensions in Firefox suffer from a different bug that makes even easier to detect the installed extensions. The browser raises an exception if a webpage requests a resource for non-installed extension (case A in Figure 7.4), but not in the case when the resource path does not exist (case B in Figure 7.4). While the exception does not cause any visible effect in the page, an attacker can simply encapsulate the invocation in a `try-catch` block to distinguish between the two execution paths and reliably test for the presence of a given extension.

By telling apart the two centralized checks that are part of the extension settings validation (either because of the side-channel or because of the different exception behaviors), it is possible to completely enumerate all the installed extensions. It is sufficient for an attacker to simply probe in a loop all existing extensions to precisely enumerate the ones installed in the system.

Table 7.1: Percentage extension detected by previous methods.

	Chrome	**Firefox**	**Total**
# Extensions Tested	10,620	10,620	21,240
% Previous Approaches	12.73%	8.17%	10.45%
% Our Approach	*100.00%*	*100.00%*	*100.00%*

In comparison, previous bypassing techniques [145, 42] were only able to detect a small subset of the existing extensions. In order to precisely assess the accuracy improvement over these previous techniques, we conducted an experiment on a set of 21,240 extensions. For this test, we decided to focus on the two browsers with the highest number of available extensions: Chrome and Firefox (Opera also has its own extension store, but the number of popular extensions is very low compared with the other browsers). In the case of Chrome, extensions are divided in three different groups: extensions, apps, and games. Although one of the groups is explicitly called extensions, all of them are installed as a `chrome-extension` and follow the same access control settings model.

At the time of writing, the number of recommended extensions in the games category (the smallest of the three) was 3,540. To keep a balanced dataset, we therefore selected also the top 3,540 of the remaining two categories, resulting in a balanced dataset of the 10,620 most recommended extensions.

For Firefox, the selection process was easier because its store makes no distinction among different categories. Therefore, we selected the 10,620 most popular Firefox extensions to keep our complete dataset equally balanced between the two browsers.

To measure the coverage of previous bypassing methods and compare it with the full coverage of our bypass technique, we combined the methods described in [145, 42]. These methods are, to the best of our knowledge, the only ones that exist capable of enumerating extensions by subverting access control settings. These methods are based on checking the existence of externally accessible resources in extensions. To test them, we analyzed the manifest files of all extensions we downloaded, looking for any accessible resources.

Table 7.1 shows the obtained coverage using previous methods. Chrome extensions were easier to enumerate than the ones in the Firefox store. However, the coverage of these old methods is very low compared to the

```
1  wot.rating = {
2  toggleframe: function(id, file, style){
3   try {
4     var frame = document.getElementById(id);
5     if (frame) {
6      frame.parentNode.removeChild(frame);
7      return true;
8     } else {
9      var body = document.getElementsByTagName("body");
10     if (body && body.length) {
11      frame = document.createElement("iframe");
12      if (frame) {
13       frame.src = safari.extension.baseURI+file;
14       frame.setAttribute("id", id);
15       frame.setAttribute("style", style);
16       if (body[0].appendChild(frame))
17        {return true;}
18     }
19    }
20   }
21  } catch (e) {
22    console.log("failed with"+e+"\n");}
23  return false;
24 }
```

Figure 7.6: Web Of Trust Safari extension function that creates an `iframe` in the website with the `baseURI` random variable as source.

full coverage achieved by our method.

7.2.2 URI Leakage

Even if URI randomization control is completely centralized, it strongly depends on developers to keep resources away from any third-party access. In fact, extensions are often used to inject additional content, controls, or simply alert panels into a website. This newly generated content can unintentionally leak the random extension URI, thus bypassing the security control measures and opening access to all the extension resources to any other code running in the same page. In addition, the leaked random URI may be used by third-parties to unequivocally identify the user while browsing during the same session.

A simple example taken from the Web of Trust[1] extension is shown in Figure 7.6. The code snippet creates a new `iframe` (line #11), sets its `src` attribute to the `baseURI` random address of the extension (line #14), and adds the frame to the document body (line #19). As a result, any other JavaScript code running in the same page (and therefore potentially under control of an attacker) can retrieve the address of the injected `iframe` and use it to access any resource of the extension. In fact, once the random token is known, the browser offers no other security mechanism to protect the access to an extension resources.

While this may seem like a simple bug in the extension development, our experiments show that it is instead a very widespread phenomenon. The entire security of the extension access control in Safari relies on the secrecy of the randomly generated token. However, the token is part of the extension URI which is often used by the extensions to reference public resources injected in the page. As a result, we believe that this design choice makes it very easy for developers to unintentionally leak the secret token.

Estimating the Scale of the Problem

The Web-of-Trust example discussed above consists of a single function of 30 lines of code, but not all the cases are so obvious to identify without a complex static analysis of the extension.

To estimate how prevalent the problem is, we implemented a prototype analyzer that reports candidate cases of URI leakage in all Safari extensions. Our tool is based on Esprima[2] to perform a static analysis based on the Abstract Syntax Trees (ASTs) of all the different JavaScript components of the extension under analysis. Source and sinks are located by just looking for the specific code in the nodes of the tree, while the information flow is computed by following the different pieces of code that actually have access to the data along the different execution paths. In particular, the analysis is performed in three steps:

1 In the first step, the tool identifies the *source* locations where the code accesses the random extension URI (looking for calls to the `baseURI` method).

2 The tool then separately analyzes all the components that can use the retrieved value. Following the information flow (i.e., functions that

[1] `https://www.mywot.com/`

[2] `https://github.com/jquery/esprima`

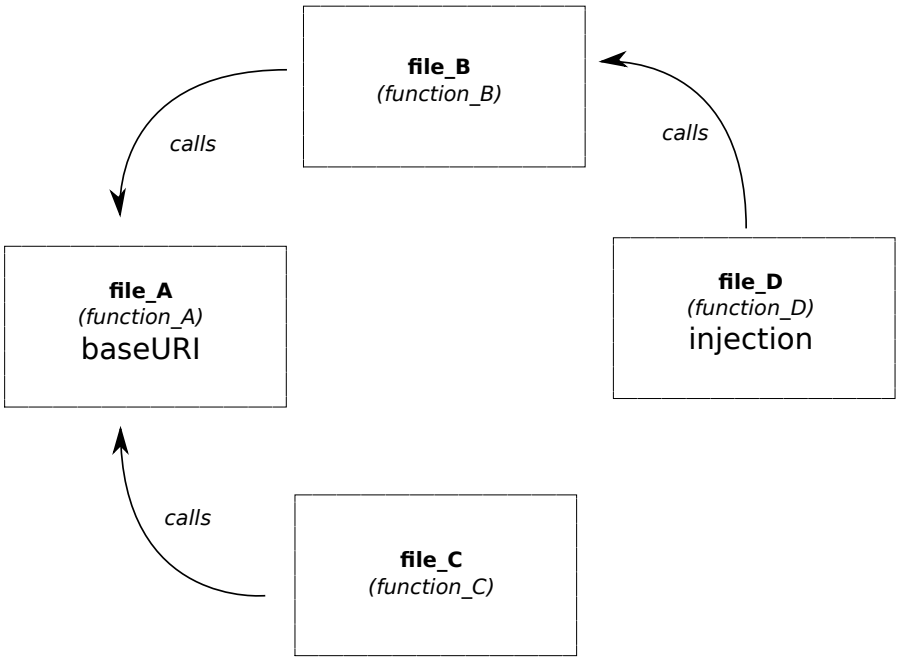


Figure 7.7: Simplified example schema of an extension that leaks the `baseURI`.

are are called or are calling), this process is performed recursively until no more connections are found.

3 For every identified components, the tool locates the *sinks*, i.e., the location where new content is injected in the webpage (e.g., through the `createElement` and `appendChild` methods). If there is a connection between the `baseURI` access and the injection of an element in the website, the extension is flagged as suspicious and reported for further analysis.

The schema in Figure 7.7 shows a simplified example of an extension that leaks the `baseURI` using `function_A` of `file_A` to obtain the value, `function_B` of `file_B` as an intermediate phase, and `function_D` of `file_D` to finally make the injection on the website.

This technique is designed to act as a screening filter and NOT as a precise detection method. Indeed, the fact that an extension retrieves the `baseURI` and then uses it to create some content is not sufficient to identify if the full information is actually leaked. For instance, we found an extension that used the `baseURI` to retrieve its version number and then injected an `iframe` with the version number included directly as part of its URL, but without leaking the complete `baseURI`.

Table 7.2: Percentage of potential baseURI leakage in safari extensions.

Category	# Total Ext.	# P. Leak
Shopping	95	57.89%
Email	13	53.85%
Security	84	52.38%
News	20	45.00%
Photos	25	44.00%
Bookmarking	61	42.62%
Productivity	147	40.82%
RSStools	5	40.00%
Entertainment	37	37.84%
Translation	8	37.50%
Social	80	30.00%
Developer	57	29.82%
Other	42	26.19%
Search	42	21.43%
urlshorteners	5	0.00%
Total	*721*	*40.50%*

To evaluate our tool, we downloaded and analyzed all the available extensions within the Safari Extension Gallery[1]. The 718 extensions belonged to 15 different categories (e.g., security, shopping, news, social networking, and search tools).

Table 7.2 shows the obtained results. In general, more than 40% of the Safari Extension Gallery were potentially vulnerable to our enumeration technique. We decided to manually analyze some of the results to determine whether the reported extensions actually performed the leak or not. Since the security category is among the ones with the highest percentage of extensions with a potential leak and it is also particularly sensitive due to the type of information these extensions usually deal with (such as user passwords), we decided to manually verify all the results for the extensions in this category.

With a considerable effort, we performed an exhaustive manual code review of all the security extensions, selecting those that were completely functional, excluding the ones that required payment for their services. Among the 68 extensions in this group, 29 were flagged as suspicious of making the leakage and 39 were not leaking it. From the suspicious ones, 20 out of 29 actually leaked the secret `baseURI`. In addition, we only identified one false negative that leaked the information but was not identified by

[1] `https://extensions.apple.com/`

our static analysis tool. In particular, this extension obtained the complete `URL`, including `baseURI`, but stored it locally. Within the extensions that are vulnerable to our attack, we found popular protection extensions such as Adblock[1], Ghostery[2], Web Of Trust[3], and Adguard[4]. The list also includes password managers, such as LastPass[5], Dashline[6], Keeper[7], and TeedyID[8] and combinations of the two functionalities (e.g., Blur from Abine[9]).

In summary, a relevant number of Safari extensions are vulnerable to our technique, including important and very popular security-related extensions. As explained in §7.4, we are now in the process of validating all the results and contacting the developers of the affected extensions to fix their code.

7.3 Impact

In the previous section we discussed the security of access control settings and URI randomization, and we showed how every mechanism adopted by current browsers can be easily bypassed in practice. There are several possible consequences of abusing the information provided by our two techniques.

7.3.1 Fingerprinting & Analytics

The most accurate and controversial form of fingerprinting aims at building a unique identifier for each user device, such as *Panopticlick* [79]. It is considered a *stateless* technique, because in order to build and share the unique identifier, these techniques do not require to store anything on the user machine (in contrast with *stateful* techniques such as *Cookies*). To build a unique identifier, several features are retrieved from the user's machine and combined in a unique fingerprint. This process can be repeated across multiple websites and the identifier will always be the same for the same

[1] `https://getadblock.com/`
[2] `https://www.ghostery.com/`
[3] `https://www.mywot.com/`
[4] `https://adguard.com/`
[5] `https://lastpass.com/`
[6] `https://www.dashlane.com`
[7] `https://keepersecurity.com/`
[8] `https://www.teddyid.com/`
[9] `https://dnt.abine.com`

machine, allowing trackers to determine users' browsing history, among other tasks. Using the set of installed extensions can increase the uniqueness of the resulting fingerprint. To measure the exact fingerprinting ability of extension enumerations, a study should be performed to measure the discriminatory power of the most popular extensions available for each browser. To this end, we have conducted a preliminary study of this type of analysis in §7.3.3.

The techniques proposed in this chapter can also be used to perform a completely accurate browser fingerprinting without checking the `User-Agent`. To this end, our method can be used to check for **built-in extensions**. These extensions are pre-installed and present in nearly every major web browser and there is no possibility for the user to uninstall them. Therefore, if we configure our techniques to check one of these built-in extensions that does not exist in other browsers, a website can precisely identify the browser family with 100% accuracy.

The installed extensions enumeration combined with the mentioned browser identification can be used to determine users' demographics. The extensions that a particular user utilizes can be easily discovered by websites or third-party services. Installed extensions provide information of a particular user's interests, concerns, and browsing habits. For example, users with security and privacy extensions installed in their browsers such as *Ghostery* or *PrivacyBadger* are potentially more aware about their privacy than other users. The same happens with personalizing extensions, games, or any possible combinations of other extensions categories. In order to measure the feasibility of performing analytics through extensions, we have conducted a proof-concept test described in §7.3.3.

7.3.2 Malicious Applications

The information retrieved from the installed extensions can also be used for malicious purposes, as the information gathering phase about potential victims is usually the first step to perform a targeted attack. For instance, attackers can inject the extension enumeration code in a compromised website and search for users with shopping management extensions and password managers to narrow down their attack surface to only those users whose credit card information has a higher likelihood to be stolen. Another possibility would be to identify the presence of a major antivirus vendor extension to personalize an exploit kit or to decide whether the ma-

licious payload should be delivered or not to a certain user.

In addition to the attacks presented, in a recent work, Buyukkayhan et al. [43] presented *CrossFire*, a technique that allows attacker to perform malicious actions using legitimate extensions. The part that was left unanswered by the paper is how the attacker can identify a set of installed extensions to use for her purpose. By using our enumeration technique, an attacker can create completely functional malicious extensions by knowing all installed victim's extensions in beforehand.

Due to the variability of possible extensions, the information of a particular user can be exploited in many different social-driven attacks (automated or not). For example, a malicious website can exploit the information about particular extensions being installed to impersonate and fake legitimate messages about that extensions, with the intention of deceiving the user and leading her to install malicious software. As a particular example, if a malicious website discovers the user is using a particular password management extension, it can create a fake window to ask the user to retype her password. This attack is particularly severe in the case of Safari, since the attacker can actually access all the resources of an extension that leaks its `baseURI`. Therefore, even a careful user who decides to analyze the website source cannot easily understand if a certain window or frame is created by an installed extension or by the website reusing the extension resources.

While the URI randomization control bypass does not provide a complete enumeration capability, when an extension leaks its random token it opens all its internal resources to the attacker. This is potentially very harmful as it increases the attack surface, allowing the attacker to access and exploit any potential vulnerability in one of the internal extension components. For example, Kotowicz and Osborn [156] presented a Chrome extension exploitation framework[1] that could be used when it was still possible to access all the different extension resources.

7.3.3 **Viability Study**

We have studied the viability of the estimated impact for several of the cases discussed before. In particular, we have analyzed their potential for performing analytics as well as the fingerprinting capability of extensions.

[1] `https://github.com/koto/xsschef`

Table 7.3: Top 10 most Popular Extension Categories in the Chrome Store.

Category	% Usage
productivity	29.90
fun	10.45
communication	9.76
web development	7.74
accessibility	4.65
search tools	4.44
shopping	3.46
photos	3.12
news	2.40
sports	1.80

We have omitted the malicious case studies due to their inherent ethical concerns. In addition, we believe that their implementations are more straightforward than in the proof-of-concept cases we tested and evaluated.

Analytics

In the case of the analytics capability of extensions, we have computed the popularity of the different categories established in the Chrome Web Store for each of the extensions that we previously analyzed in §7.2.1. In particular, we analyzed the 63 categories present in the 10,620 most popular Chrome extensions (Table 7.3 shows the 10 most popular categories).

The most popular category was "productivity" with 29.90% usage. Nevertheless, the definition of this category is not clear because it includes a wide-range of types of extensions such as ad blockers, schedulers, or office-related tools. Anyhow, a possible sub-categorization may be possible by means of the available description of each extension. The rest of the 10 most popular are more precise and may be helpful in order to perform analytics related tasks such as targeted advertisement or website personalization. For instance, the number of visitors with "shopping", "web development", or "sports" extensions, may help the website owner to personalize her content or ads accordingly, thus improving her number of visitors or ad revenues.

However, not only the most popular extensions may help the website owner to get a better understanding of her visitors and act accordingly. Indeed, less popular extensions, because their higher power of discrimination among users, can also be used for this task. For example, the usage of exten-

sions from the "creative tools" category indicates that the visitor is prone to create content, the presence of extensions within "academic resources" category would likely indicate that the visitor is near the academic environment, "teacher tools" may imply that the visitor deliver at least some lectures, and "blogging" implies that the visitor is a blogger.

In summary, we believe that extensions are a powerful tool to perform fine-grained user analytics because of their diversity. Moreover, the information derived from the installed extensions of a web visitor, combined with the classical analytics information may lead to a better user analytics for website owners.

In order to understand and measure the capability of extensions for device fingerprinting, we implemented a page that checks the users' installed extensions among the top 1,000 most popular from the Chrome Web Store and the Add-ons Firefox websites, using the timing side-channel extension enumeration attack described in §7.2.1. Since our study involved the enumeration of several users' installed extensions, we informed the users about the procedure including the information gathered. Only after the user agrees to perform the experiment and share the collected information, the enumeration of her extensions is conducted. We also set a cookie on the user browser to prevent multiple resubmissions from the same user. In addition, to protect the user privacy, we only collected anonymous data.

We disseminate the URL through social networks and friends, asking them to participate in the study and further re-disseminate the link among their contacts. This way we collected the list of installed extensions from 204 participants from 16 different countries. Even though this number is smaller than in previous studies, we would like to remark that fingerprinting is not the actual goal of the chapter but just a possible application of our attacks. In fact, this analysis is simply designed to determine the viability of our technique for device fingerprinting, either as a method by itself or by complementing other existing fingerprinting techniques.

Device Fingerprinting

Following the standard adopted in previous works [79, 162], we analyzed the extension anonymity sets of the fingerprinted users, which is defined as the number of users with the same fingerprint i.e., same extension set (the distribution of anonymity sets is shown in Figure 7.8). Overall, from the 204 users that participated in our study, 116 users presented a unique set of installed extensions, which means that 56.86% of the participants are

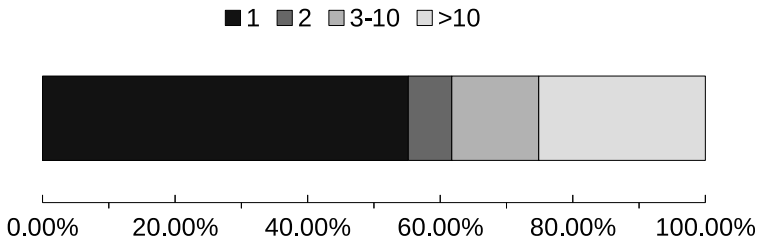


Figure 7.8: Distribution of anonymity set sizes regarding extensions.

Table 7.4: Comparison between Extensions with other Fingerprinting Attributes.

Method	Entropy
Extensions	0.869
List of Plugins	0.718
List of Fonts	0.548
User Agent	0.550
Canvas	0.475
Content Language	0.344
Screen Resolution	0.263

uniquely identifiable just by using their set of extensions.

In addition, we also compare the discriminatory level of this proof-of-concept fingerprinting technique by computing its normalized Shannon Entropy [162] and comparing it with other fingerprinting attributes proposed in previous studies. In particular, Table 7.4 compares the different entropy values of the top six fingerprinting methods or attributes measured in the work by Laperdrix et al. [162] with our extensions-based fingerprinting method. We can notice that extensions presented the highest entropy of the analyzed fingerprinting attributes — making them more precise than using the list of fonts or canvas-based techniques.

7.4 Vulnerability Disclosure and Countermeasures

7.4.1 Attack Coverage & Effects

In this chapter we presented two different classes of attacks against the resource control policies adopted by all families of browsers on the market.

Table 7.5: Current Browsers affected by our attacks. The last two lines refer to Extensions still under development.

Browser	Extensions Enumeration	Resource Access
Chromium Family	✓	
– Chrome	✓	
– Opera	✓	
– Yandex	✓	
…	✓	
Firefox Family	✓	
– Firefox Mobile	✓	
– Iceweasel	✓	
– Pale Moon	✓	
…	✓	
Safari	≤ 40%	≤ 40%
Microsoft Edge	in discussion	
Firefox WebExtensions	in discussion	

Table 7.5 summarizes the overall impact of our methods.

As already mentioned, the coverage of our enumeration attack is complete in the case of the timing side-channel attack to access-control-based browser families (i.e., Chromium and Firefox Families) while nearly around 40% in URL randomization browsers (Safari).

Effects of Private Mode

"Incognito" or private mode is present in most of the modern browsers and it protects and restricts several accesses to the browser resources such as cookies or browsing history. Therefore, we decided to analyze if our attacks can enumerate extensions even when this mode is activated.

We discovered that all of our attacks accurately identified the list of installed extensions also within the private mode. This fact is due to several reasons. In the case of Chromium family browser, the browser checks for extensions in incognito mode, even though extensions are not allowed to access the websites [53]. Firefox and Safari did not include any check for extensions and, therefore, both websites and extensions are able to access each other.

```
GetFlagsFromPackage(const nsCString& aPackage,uint32_t* aFlags){
  PackageEntry* entry;
  if (!mPackagesHash.Get(aPackage, &entry))
    return NS_ERROR_FILE_NOT_FOUND;
  *aFlags = entry->flags;
  return NS_OK;
}

GetSubstitutionInternal(const nsACString& root, nsIURI **result){
    nsAutoCString uri;
    if (!ResolveSpecialCases(root, NS_LITERAL_CSTRING("/"), uri)) {
        return NS_ERROR_NOT_AVAILABLE;}
    return NS_NewURI(result, uri);
}
```

Figure 7.9: Firefox functions that cause the difference between existing and not existing extensions.

```
const Extension* extension=RendererExtensionRegistry::
Get()->GetExtensionOrAppByURL(resource_url);
if (!extension) {
  return true;
}
```

Figure 7.10: Snip of Resource Request Policy function of Chromium that causes the difference between existing and not existing extensions (see Appendix for full code).

7.4.2 Timing Side-Channel Attack

The first class of attacks is the consequence of a poor implementation of the browser access control settings: Firefox-family browsers usage of extensions can be exploited to recognize the reason behind a failed resolution, and Chromium family timing-side channel allows an attacker to precisely tell apart the two individual checks performed by the browser engine.

The consequence, in both cases, is a perfect technique to enumerate all the extensions installed by the user. Given the open-source nature of these two browsers, we manually identified the functions responsible of the problem and indicated how to fix each of them.

Chromium family

We contacted the Chromium team to report the timing problem. The developers were quite surprised about the attack, because they believed that the time differences in the checking phase were not significant enough to allow this type of timing side-channel attack. By inspecting the function responsible of checking the accessibility of a concrete extension path (see Figure 7.10), the two different steps described in section 7.2 can be clearly identified. First, the browser tests the existence of the extension (line #4) and finishes if the extension does not exist. If the extension does exist, it performs different checks to make sure that the path is accessible, returning a error message if it is not. These checks are the ones that permit the timing difference exploited in the attack.

We suggested a possible way to fix the code to avoid the time measurement by modifying the extension control mechanism to combine the internal extension verification and the resource check together in a single atomic operation (i.e., by modifying the extension existence check of line #4). This requires to replace the extension list with a hashtable containing the extensions and the full path of their resources.

While it may seem simple to fix by making the check atomic, the problem remains if the attack is performed with real extension paths (easily obtainable) instead of fake paths. The timing difference would be the same as the one presented in Figure 7.4, with the only difference that the first check would validate the full path and not just the extension. At the time of writing, as it is a design-related problem, it is still not fixed.

In addition, as the new Firefox WebExtensions and Microsoft Edge (both currently in their early stages) use the same extension control mechanisms proposed by Chromium, we also notified their developers to make them aware of the issue described in this chapter. We hope that our effort will help these two new versions to integrate by-design the necessary countermeasures to avoid these security problems since the beginning.

Firefox family

We also responsibly reported the Firefox non-WebExtensions problem that makes our enumeration attack possible to its developers, who acknowledged the issue and are currently discussing how to proceed. Specifically, Figure 7.9 show the function that causes the response difference regarding the extension existence.

The error returned when the resource path does not exist (line #4 and line #12 in Figure 7.9) does not raise any exception. Therefore, the solution is straightforward: return a `NS_ERROR_DOM_BAD_URI` error (i.e., the same one that is thrown when extension is not installed). This fix will not cause any issue to websites using extension paths, maintaining the functionality intact.

Regarding WebExtensions, the Firefox developers recently changed the way extensions are accessed in order to solve the timing side-channel and other related attacks. In particular, they changed the initial scheme (`moz-extension://[extID]/[path]`) to the new implemented version of `moz-extension://[random-UUID]/[path]`. Unfortunately, while this change makes indeed more difficult to enumerate user extensions, it introduces a far more dangerous problem. In fact, the `random-UUID` token can now be used to precisely fingerprint users if it is leaked by an extensions. A website can retrieve this UUID and use it to uniquely identify the user, as once it is generated the random ID never changes. We reported this design-related bug to Firefox developers as well.

7.4.3 URI Leakage

The second class of attacks presented in the chapter is quite different. In fact, the method that Safari's extension control employs to assure the correct accessibility of resources is, in principle, correct. However, Safari delegates to the extension developers the responsibility to keep the random URI secret. We believe that this is a very risky decision because most of the developers lack a proper understanding of the problem. As a consequence, our experiments confirm that a relevant number (40% in our preliminary experiments) of the extensions are likely to leak the `baseURI`, undermining the entire security solution. In particular, we discovered that important security extensions such as multiple password managers or advertisement blockers suffer from this `baseURI` leakage vulnerability and, hence, they are vulnerable to this attack. In the case of security extensions, this is particularly worrying due to the type of information they manage is usually very sensitive.

In this case the problem is even harder to solve, because it is not a consequence of an error in the extension control but of hundreds of errors spread over different extensions. Reaching out and training all the extension developers is a difficult task but Apple should provide more information on the proper way to handle the `baseURI` and about the security implications

of this process.

In addition, we believe that Safari could benefit from adopting a basic lightweight static analysis solution (similar to the one we discuss in §7.2) to analyze the extensions in their market and flag those that leak the random token. This would allow to immediately identify potentially leaking extensions that may need a more accurate manual verification. In the meantime, we started reporting the problem to some security extensions we already manually confirmed, to help them solve their URI leakage problem.

7.4.4 Extension Security Proposal

In order to improve the security and privacy of browser extensions, we propose a solution that solves all the different problems presented in this chapter.

1 All browsers should follow an extension schema that includes a random generated value: `X-extension://[randomValue]/[path]`. This random value should be modified across and during the same session and should be independent for each extension installed. For example, the browser should change it in every extension in every access. In this way, the random value cannot be used to fingerprint users.

2 Browsers should also implement an access control to avoid any undesirable access to all extensions resources even when the random value is unintentionally leaked (such as `web_accessible_resource`) .

3 Extensions should be analyzed for possible leakages before making them public to the users. Moreover, developer manuals should specifically discuss the problems that can cause the leakage of any random value generated.

7.5 Related Work

Security of Browser Extensions

The research community has made a large number of contributions analyzing the security properties of browsers extensions. A number of recent studies have focused on monitoring the runtime execution of browser extensions. Louw et al. [269, 270] proposed an integrity checker and a policy

enforcement for Firefox legacy extensions. A more recent framework, Sentinel [218, 219], provided a fine-grained control to the users over legacy extensions, allowing them to define custom security policies while blocking common attacks to these extensions.

Other approaches have focused on security analysis of browsers extensions in order to discover security flaws. On the static analysis side, IBEX [125] is a framework to analyze security properties by means of a static methodology and it also allows developers to create a fine-grained access control and data-flow policies. VEX [29] is instead a static analyzer for Firefox JavaScript extensions that applies information flow analysis to identify browser extension vulnerabilities.

Dynamic extensions analysis includes the work of Djeric et al. [77], in which the authors proposed the use of dynamic analysis to track data inside the browser and detect malicious extensions. Dhawan et al. [72] proposed a similar approach to detect extensions that compromised the browser environment. In a similar vein, Wang et al. [280] used an instrumented browser to analyze Firefox Extensions. Hulk [148] is a dynamic analysis framework that controlled the activity of the browsing extensions, employing fuzzing techniques and HoneyPages adapted to the extensions. Hulk was used to analyze more than 48,000 Chrome extensions, discovering several malicious ones.

Despite the fact that these approaches are useful to detect malicious or compromised extensions, they are unfortunately useless against external attacks or information leakages. Our analysis has lead to the most complete set of attacks against resource accessibility control and `baseURI` randomization, allowing in both cases extension enumeration attacks that can be used as part of larger threats.

Similar to our own work, XHOUND [260] recently showed that the changes extensions perform on the DOM are enough to enumerate extensions. Using this technique, the authors also developed a new device fingerprinting technique and measured its impact. However, this approach has a much more limited applicability. In comparison, our techniques achieve a larger coverage, successfully enumerating 100% of the extensions for *access control* browsers and around 40% for those using *URI randomization*.

Web Timing Attacks

Web Timing attacks have been used for many different purposes, both in the client side and server side. Felten and Schneider [92] introduced this

type of attacks as a tool to compromise users' private data and, specifically, their web-browsing history. In this way, a malicious attacker might obtain this information by leveraging the different forms of web browser cache techniques. By measuring the time needed to access certain data from an unrelated website, the researchers could determine if that specific data was cached or not, indicating a previous access.

Later, Bortz et al. [38] organized timing attacks in two different types of attacks: (i) direct timing, consisting in measuring the time difference in HTTP requests to websites and (ii) cross-site timing, which allows to obtain data from the client-side. The first type could expose website data that may be used to prove the validity of a username in certain secure website. The second type of attacks follow the same line of work of previous work by Felten and Schneider. They also performed some experiments that suggested that these timing vulnerabilities were more common than expected. In addition, Kotcher et al. [155] discovered that besides from the attacks previous discussed, the usage of CSS filters made possible the revelation of sensitive information such as text tokens exploiting time differences to render various DOM trees.

Two recent studies show that these attacks are far from being solved. Jia et al. [144] analyzed the possibility of determining the geo-locations of users thanks to the customization of services performed by websites. Location-sensitive content is cached the same way as any other content. Therefore, a malicious actor can determine the victim's location by checking this concrete data and without relying in any other technique. Besides, Van Goethem et al. [276] proposed new timing techniques based on estimating the size of cross-origin resources. Since the measurement starts after the resources are downloaded, it does not suffer from unfavorable network conditions. The study also shows that these attacks could be used in various platforms, increasing the attack surface and the number of potential victims.

However, none of these timing techniques have been previously used to identify components of the web browser itself. Our new timing side-channel attacks are the first attacks capable of determining with 100% accuracy which extensions are installed in the browser, independently of the CPU usage.

7.6 Conclusions

Many different threats against the users security and privacy can benefit from a precise fingerprint of the extensions installed in the browser.

In this chapter, we show that the current countermeasures adopted by all browser families are insufficient or erroneously implemented. In particular, we present a novel time side-channel attack against the access control settings used by the Chromium browser family. This technique is capable of correctly identifying any installed extension. Firefox WebExtensions and Microsoft Edge (early states) follow the same API and design, indicating that they may be prone to be vulnerable to the attack.

We also discuss a URI leakage technique that subverts the URI randomization mechanism implemented in Safari, that emerges from inappropriate extension implementations that leak the value of a random token. We implemented a new method to identify extensions with this potential leakage and we found out that up to 40% of Safari extensions could be vulnerable to this problem. After a manual inspection of security-related extensions, we discovered that many popular extensions are vulnerable to this attack. In addition, in the case of this attack, not only the extension is identified but also its resources can be accessed, posing as a more dangerous threat.

We also presented applications for our extension enumeration attacks. First, we propose different fingerprinting and user analytics techniques, demonstrating their feasibility in a real-world scenario. Second, we also proposed technique to use our enumeration techniques for malicious applications such as targeted malware, social engineering, or vulnerable extension exploitation.

We responsibly disclosed all our findings and we are now discussing with the developers of several browsers and extensions to propose the correct countermeasures to mitigate these attacks in both current and future versions.

"I sat down one night and wrote the line rock, rock, rock everybody."

Bill Haley (1925–1981)

Chapter 8

Clock Around the Clock: Time-Based Device Fingerprinting

A large number of *physical device fingerprinting* techniques have been proposed over the years to uniquely identify a device based on its physical features [37, 39, 212, 223, 112, 165]. The application of these techniques also varies, and includes device authentication, software license binding, online attackers identification [154, 96], and wireless network identification [30, 107].

More recently, hardware-level features have also been adopted to create more precise forms of web tracking. In what is normally called *web device fingerprinting*, the owner of a website computes a unique identifier for each visitor's machine, without storing any information on the client side — thus making these techniques easier to hide and harder to block or mitigate. Its stateless nature is what makes device fingerprinting particularly relevant for web tracking. Since the user's unique identification is computed every time she visits a website, it is not possible for the user to remove the fingerprint, making this more difficult to avoid than older stateful web tracking approaches. We can distinguish between two types of device fingerprinting techniques: we refer to those that are based on browser artifacts as *attribute-based device fingerprinting* and to those based on hardware-

level features as *hardware-level device fingerprinting*. Attribute-based techniques relies on different accessible browser attributes such as the list of installed fonts, the `UserAgent` string, and the screen resolution. Since these attributes change often and are easy for the user to modify, the resulting fingerprint also rapidly evolves – thus preventing a stable, long-lasting tracking [278]. In contrast, hardware-level techniques exploit subtle differences in the underlying hardware that are detectable by invoking certain APIs to compute the differences between devices. For instance, it is possible to compute differences in the way text is rendered by the HTML5 Canvas API or by using the WebGL API [199]. Even though these techniques are very promising and less prone to periodic changes than attribute-based solutions, all the hardware-based techniques proposed to date depend not only on the hardware itself, but also on the particular APIs implementation in the target browsers.

In this chapter, we propose to look at code execution time as a way to precisely identify different devices. The time a computer spends to execute an instruction depends on how many clock cycles the instruction requires, and on the duration of each cycle. Internal clocks use oscillators based on quartz crystals, and small variations in those crystals can result in extremely small, but measurable, differences in the clock frequency. Researchers have already proposed to use these differences to uniquely fingerprint different devices [154, 227], but previous measurements were difficult to take, as they needed to analyze network traffic, and required an external reference time to compare with. Salo [237] proposed a solution to this problem by comparing two different clocks: the one used by the CPU and the independent one used to maintain the internal timer. However, the proposed methodology strongly depends on specific hardware, relied on custom snippet of assembly code, and required a long execution time to generate a stable fingerprint.

Our idea relies instead in the identification of readily available functions that, when repeated a sufficient number of times, can be used to amplify the small differences between different clocks. Those functions should contain enough instructions to achieve a sufficient precision, but not too many to be regularly interrupted by the OS scheduler. We then measure the execution time by using the `datetime` APIs, which rely on a separate clock than the one used by the CPU to execute code. Our experiments show that this approach can be used to precisely fingerprint a machine, even when performed by using a snippet of JavaScript embedded in a web page. After testing with a set of candidate functions, we settled our proof

of concept implementation on a simple cryptographic routine to generate pseudo-random numbers, as it is widely available and it is commonly used as basic block in many popular applications.

Our experiments demonstrate that subtle differences in the execution times of this cryptographic function are sufficient to capture the differences among different machines, outperforming all hardware device fingerprinting techniques proposed to date. To obtain a baseline to compare the fingerprinting capabilities of our approach, we first implemented a native version of our method in C. The tool was stress-tested in a scenario with hundreds of computers equipped with the exact same hardware and software. Then, in order to verify that our solution can also be used for web device fingerprinting, we implemented a web version of our algorithm using the HTML5 Cryptography API — that ultimately invokes the same operating systems functionalities we relied upon in the C version. This web implementation was tested using the same homogeneous scenario composed of computers with same software and hardware configuration as well as a real-world scenario including different users who visited our public experiment website, making a total of 565 different users.

The remainder of the chapter is organized as follows. §8.1 proposes a new set of features for assessing fingerprinting methods, tackling the current limitations regarding fingerprinting methods evaluation. §8.2 details the proposed hardware machine fingerprinting method, explaining the reasons that make these new techniques to work accurately. §8.3 evaluates the native method in a homogeneous scenario environment, showing that it can discriminate between identical hardware machines. §8.4 details the specific implementation in the web, for a web device fingerprinting technique, evaluating this method and comparing it with current state-of-the-art in hardware-level web device fingerprinting techniques, both in a homogeneous scenario environment and in an in-the-wild real world scenario experiment. §9.5 discusses the major implications of this work. §9.6 provides the reader with the required background on device fingerprinting and timing attacks, critically analyzing existing methods. Finally, §9.7 provides the concluding remarks.

8.1 **Fingerprint Assessment**

The goal of fingerprinting techniques is to uniquely identify a target entity. This entity can be a browser, a physical machine, or even a user across

different personal devices. Despite the large number of fingerprinting techniques proposed by both academia and industry (see §9.6 for more details) — or already discovered in the wild, the security community has not come yet to a consensus on which characteristics need to be measured to properly evaluate and compare between fingerprinting solutions.

For instance, for web-based fingerprinting approaches, the *cross-entropy* and the size of the *anonymity set* are used as the "de facto" evaluation standard procedure across multiple papers [79, 162, 48]. While certainly important, they fail to capture many important aspects of a fingerprinting procedure, such as its resilience to changes in the user browser (e.g., due to a software update) or the overall efficiency of the computation process.

In this chapter, we propose a rich set of metrics to be used as a new basis to measure the quality of different fingerprints. This set includes six "desired" characteristics of a fingerprint:

- **Discrimination Power:** The discrimination power of a fingerprint is defined as its ability to produce different fingerprints for different targets. This can measure the ability to uniquely identify a target among a set of possible candidates.
- **Stability:** The stability of a fingerprinting technique is its ability to always produce the same fingerprint for the same target over multiple measurements.
- **Homogeneous Discrimination:** This property measures the discrimination power of a technique in the case in which the targets belong to the same homogeneous family, and they are therefore similar among each other.
- **Efficiency:** This feature simply measures the time required to generate the fingerprints and check them against a database of previous candidates.
- **Resilience to Evasion:** Since there exist methods to avoid fingerprinting or at least to reduce its consequences, this feature takes into account whether a fingerprinting technique is resilient to known or possible evasions.
- **Resilience to Change:** This final characteristic captures the ability of a fingerprinting technique to remain stable over time. Some techniques use features that naturally evolve, thus resulting in a finger-

print that can be associated to a target only for a limited time window. Indeed, a recent paper [278] has studied the evolution of existing techniques and has found that the vast majority of the general fingerprints changes in less than 10 days.

Unfortunately, current evaluation procedures usually assess only the discrimination power (by measuring the cross-entropy and the anonymity sets) and sometimes the efficiency of a solution. However, we believe that all features are equally important to comprehensively assess a new fingerprinting technique.

In particular, the stability has been surprisingly omitted from many evaluations presented to date. If a fingerprint lacks stability, it means that the procedure may generate erroneous fingerprints or that the result includes a certain level of noise, misleading the identification. Please note that stability should not be mistaken with resilience to change. The latter deals with the natural fingerprint evolution over time, rather than the fact that a technique may return different values for the same device when executed multiple times.

In a similar vein, the most common way researchers used to compile a test set for a new fingerprinting method and to compute its discriminatory power is to host it on a web page and share its URL with a large number of users. In this case, the number of different configurations both in hardware and software is high, even more if we consider that most of the machines will be commodity user computers. This setting results in some attributes, such as the `UserAgent`, to show a very high cross-entropy [162], against intuitive observations. A more homogeneous environment (e.g., the set of many similar or identical computers that form many company networks) would provide a much more challenging environment to assess the fingerprinting precision.

Finally, with the notable exception of Cao et al. [48], no fingerprinting has taken into account the error introduced by possible changes in the user browser or operating system. This is another fundamental aspect of the problem, as even the most accurate solutions is of limited use if the fingerprints changes every time a user reboots the computer or installs a software update.

To better understand the importance of these metrics, we reviewed the characteristics of a number of current state-of-the-art fingerprinting methods, namely (i) Attribute-based FP, (ii) CanvasFP, (iii) WebGL FP and (iv) AudioFP. It is important to remark that this comparison is only based on

what has already been tested by the original authors (in the particular case of WebGL, we conducted experiments using the available open-source tool) or based on the design of the fingerprinting method.

Interestingly, none of the methods described so far is resilient to changes in the target environment — with the exception of the aforementioned work by Cao et al. [48]. For instance, a simple graphic driver update can completely modify the fingerprint obtained by CanvasFP or WebGL. Evasion can also be easily implemented for all the methods, and actually most of them are already completely ineffective against the Tor Browser. According to the results presented in their respective papers, all techniques are capable of discriminating different targets (note that we have grouped attribute-based fingerprinting together as they are usually utilized altogether). WebGL was poor on the efficiency axis, as the version we tested required several seconds to build a single fingerprint. Unfortunately, the homogeneous discrimination and stability are difficult to estimate. Since we consider all of these features equally important, we will compare our method and the state-of-the-art hardware-level fingerprinting methods in all these dimensions in §9.6.

8.2 Host Fingerprinting based on Clock Imperfections

In this section we present a new machine fingerprinting technique based on timing the execution of several invocations, performed using different parameters, of a properly selected function. The main assumption behind our solution is that it is possible to measure small variations on the execution time of a sufficiently long sequence of instructions that are introduced by imprecisions and imperfections (also known as "process variation" in the VLSI and architecture communities) of the machine clock crystal .

8.2.1 Threat Model and Use Cases

We test our time-based fingerprinting method in two different yet complementary scenarios. In the first one, we implement our technique in C and use it to tune our algorithm, while providing a baseline for comparison for the remote scenario. In the second use case, we port our technique to the web device fingerprinting scenario, implementing it in HTML5 and, thus, testing its ability to fingerprint machines over the web.

Host-based Fingerprinting

In this first scenario (detailed and tested in §8.3), we test the accuracy of our time-based device fingerprinting technique running natively in the target operating system. We perform this test even when it is known that you can fingerprint the clock natively to show the capabilities of our method and provide a baseline for the web version. Here, we assume the entity interested in computing the fingerprint is able to run arbitrary code with user privileges in the physical machine. For instance, this is the case of (i) malicious applications that want this information to perform selective attacks against certain victims, and (ii) proprietary applications that want to bind a license to a single machine.

Web-based Fingerprinting

The second scenario is more challenging, as we imagine that the entity who wants to compute the fingerprint is now an arbitrary website containing JavaScript code. In this case, it is not possible to run arbitrary instructions on the CPU, as modern browsers introduce numerous intermediate layers between the JavaScript code and the final CPU instructions. The goal of this scenario is to test if our approach can also be remotely executed over the web, thus resulting in a very powerful new technique for device fingerprinting. Also in this case, we can envision two different scenarios: (i) advertisers or tracking companies can use it to obtain the browsing history of their users, and (ii) websites that require strong authentication (e.g., banking and shopping) can use this technique to include an additional verification to their process.

8.2.2 Existing Approach

The detection of clock imperfections for fingerprinting purposes has already been exploited on a single CPU by Salo [237], but this solution required complex native experiments (which made the technique difficult to use in the real world) and were not able to successfully discriminate all machines involved in the test. To detect the imperfections, Salo proposed to compare the CPU clock cycles of ticks in the clock with the cycles needed for the digitalization of an analog signal using the sound card (all validated by an external GPS receiver). Afterwards, the author computed different statistical tests to distinguish among different machines. Several factors play a crucial role for this technique to work:

1 The program needs to have access to the CPU clock cycles, which is not a big problem for a low-level programming language as C or C++, but is not a common option in high-level languages as JavaScript. Furthermore, some specific tuning needs to be done depending on the specific type of CPU used in the experiments.

2 The sound card used for the digitalization must not rely on the CPU clock and should use an independent crystal-controlled oscillator.

3 To obtain enough data to successfully distinguish between two or more machines, the experiment needed to run for approximately one hour.

These limitations show that the technique strongly depends on some specific hardware, tuning, and a long computation time — making the entire approach poorly usable in practice. Even when these requirements are satisfied, the method can only be used with low-level programming languages that can obtain direct control over the CPU clock cycles. Moreover, the results obtained show that despite many machines (from the 38 analyzed) could be differentiated, not all were correctly identified.

8.2.3 Our Approach: Time-Based Device Fingerprinting

We now present our approach, which takes just some milliseconds to execute, can be used both in low or high level programming languages, and is not dependent on any specific hardware. Our algorithm is divided into two different phases: the generation of the fingerprint performed by timing a given function, and the comparison phase in which we test whether a pair of fingerprints (which consists of a matrix of time results) belong to the same machine.

8.2.3.1 Fingerprint Generation

In this phase, the algorithm computes the time required to execute different invocations of a target function (see Figure 8.1 for the detailed pseudo-code of the algorithm). The algorithm takes one parameter n that indicates the number of calls to measure. Moreover, for the sake of simplicity, in the example in Figure 8.1 we have assumed that this number is also used as parameter for the function itself. For instance, if we use a function that generates random numbers, we will consecutively create different number

Input: n, number of timings to perform
Input: m, number of arrays of these timings to generate.
Output: fp, array of arrays of numbers representing the fingerprint: each position are the result of timings with a different parameter for a function.

```
Function FPGeneration (n, m)
    i ← 1  fp ← float[][] of size n × m  while i ≤ m do
        j ← 1  while j ≤ n do
            startTime ← GetCurrentTime() ;
             Function(j);
             endTime ← GetCurrentTime();
             logTime ← endTime − startTime;
             fp[j][i] ← logTime;
             j ← j + 1;
        end
        i ← i + 1;
    end
    return fp;
```

Figure 8.1: Fingerprint Generation Algorithm.

of random values, allowing us to time the functions in different situations depending on the input.

There are many factors that can cause performance variability in non-deterministic ways. Pure hardware-level factors as Cache/TLB misses and sharing the pipeline resources with other threads co-scheduled on the same core (hyper-threading) or even OS's DVFS (Dynamic Voltage Frequency Scaling) decisions. Because of all these possible non-deterministic factors, a single measure is insufficient to obtain a stable measurement. In order to obtain stable fingerprints, our method uses an additional parameter m that determines the number of times this process is repeated, to achieve a real representation of the machine independently of different specific situations. As a result, the final fingerprint is a $n * m$ matrix of execution times. To sum up, there are m function calls, with specific values as input, computed for each of the n rows of the timing matrix.

As the technique is not based on computing the same function with the same input all the time, but executing the same function with different inputs, the matrix structure allows a quick comparison with other fingerprints. For example, following the case of generating random numbers presented before, we can easily check the differences between the fifth execution of the function that generated 20 random numbers in one computer

Input: $fp1$, $1st$ array of arrays of timing results sized $n \times m$.
Input: $fp2$, $2nd$ array of arrays of timing results sized $n \times m$.
Input: n, number of timings for different parameters.
Input: m, number of arrays of timings generated.
Output: indicates the number of coincidences

Function `GetNumCoincidences` $(fp1, fp2, n, m)$
$num\_coindences \leftarrow 0$;
$fp1\_mode \leftarrow float[]$;
$i \leftarrow 1$;
while $i \leq n$ **do**
$fp1\_mode[i] \leftarrow$ `ComputeMode`$(fp1[i])$;
$i \leftarrow i + 1$;
end
$i \leftarrow 1$;
while $i \leq n$ **do**
$check \leftarrow$ `false`;
$j \leftarrow 1$;
while $(j \leq m) \wedge (\neg check)$ **do**
if $fp1\_mode[i] = fp2[i][j]$ **then**
$num\_coindences \leftarrow num\_coindences + 1$;
$check \leftarrow$ `true`;
end
else
$j \leftarrow j + 1$;
end
end
$i \leftarrow i + 1$;
end
return $num\_coincidences$

Input: $fp1$, $1st$ array of arrays of timing results sized $n \times m$.
Input: $fp2$, $2nd$ array of arrays of timing results sized $n \times m$.
Input: n, number of timings for different parameters.
Input: m, number of arrays of timings generated.
Input: t, threshold to consider the fingerprint the same
Output: indicates if $fp1$ corresponds to $fp2$

Function `FPCheck` $(fp1, fp2, m, n, t)$
$num \leftarrow$`GetNumCoincidences`$(fp1, fp2, n, m)$;
$num \leftarrow num +$ `GetNumCoincidences`$(fp2, fp1, n, m)$;
return $(\frac{num}{n \cdot 2})) \geq t)$;

Figure 8.2: Checking Algorithm.

with exactly their counterpart on another computer.

8.2.3.2 **Fingerprint Comparison**

In this phase, the system compares two previously-computed fingerprints and determines whether or not they belong to the same machine (for the detailed pseudo-code of the algorithm refer to Figure 8.2). To this end, we compute the most frequent timing values (the mode) for each call parameter over all iterations. Afterwards, the mode of the first fingerprint is compared with all the generated values for the same call in the second fingerprint. If one match is found, a counter is incremented. This process is then repeated, inverting the order and checking the most common values in the second one with all the values from the first one. If the number of matches divided by the number of comparisons surpasses a fixed threshold, then our algorithm concludes that the two fingerprints belong to the same machine.

For example, suppose we want to compare the following two fingerprints fp_1 and fp_2, each composed of three repetitions of three different timing results of the invocation of a given function:

$fp_1 = [\{0.1; 0.12; 0.14\}, \{0.1; 0.12; 0.13\}, \{0.1; 0.12; 0.13\}]$
$fp_2 = [\{0.1; 0.12; 0.14\}, \{0.11; 0.12; 0.14\}, \{0.1; 0.12; 0.13\}]$

We start generating the mode of the values of fp_1: $\{0.1; 0.12; 0.13\}$ and comparing each of the three values with the values in the three value sets of fp_2, resulting in three positive matches. The first value appears in the first and third iteration of fp_2, the second value appears in the all the iterations, and the third value appears in the last iteration. Then, we will do the same with fp_2 being their mode values: $\{0.1; 0.12; 0.14\}$ and also getting all of them matched in the fp_1 set. The first and seconds values appear in all the iterations of fp_1, and the third value appears in the first iteration. In conclusion, vectors do not need to be identical, but match each of the values of the mode with, at least, one of the value in the same position on another fingerprint. In this case, the percentage of similarity would have been 100% which, as a perfect match, would be above the threshold and our method would have determined that both fingerprints belonged to the same computer.

By using this procedure, we are computing and comparing the most common timing values — and, therefore, the most representative ones — among the measurements conducted on the two machines. This reduces

Table 8.1: Results of the Function Viability Test.

Function	Stable Fingerprint
`string::compare`	✓
`std::regex`	✓
`std::hash`	✓
`crypt`	✗

the inevitable noise introduced in the timing measurements and reduces the impact of unusual values.

8.2.3.3 Function Selection

Before settling on a final choice, we decided to perform a preliminary set of tests to assess the different candidate functions. In particular, we evaluated the functions `string::compare`, `std::regex`, `std::hash`, and `crypt`. While our technique would work also using a custom, system-independent function, we decided to base our tests on a set of common routines that can be easily found in many different systems. This increases the portability of our approach as it does not require to install or inject any additional code. The evaluation was performed on a set of ten different machines, half of which installed with Microsoft Windows and the other half installed with GNU/Linux. We also computed different tests with the aforementioned functions to empirically validate the best size of the measurement matrix, taking into account the generation time and the fingerprint discrimination capabilities. Based on these preliminary tests, we found that $n = 1000$ and $m = 50$ (i.e., a total number of 50,000 invocations) are sufficient to provide stable results.

Table 8.1 shows the obtained results. `crypt` was the only function whose fingerprint was not stable because, due to its complexity, it was often interrupted by the operating system scheduler — thus preventing our algorithm to accurately time its execution. For the remaining functions, it is important to note that simpler functions required to compute the execution time of multiple consecutive invocations to find a stable fingerprint. This issue is controllable by simply adding more iterations.

In summary, we investigated and evaluated if our fingerprinting algorithm can be built on top of multiple, diverse functions. According to our results, different candidates provided good results, in particular when they

were sufficiently complex but not too long to be often interrupted by the scheduler.

8.2.3.4 **Stability Tests**

In order to determine the viability of the proposed approach for machine fingerprinting, we conducted three additional tests. The setup for these stability tests is the same as the one used for the function selection. We checked if the obtained fingerprint of each machine can still identify the machine in the following cases:

- **CPU Load:** We tested the influence of different CPU load conditions on the fingerprint generation process. In our experiments, we controlled the CPU workload by using the `stress` generator included in the Debian distribution [114] and the corresponding tool part of Windows Sysinternals [136]. We discovered that even in the scenario of 100% CPU load, the resultant fingerprint was always correctly associated. This is a consequence of the fact that each function invocation gets executed in a single CPU with no interruption, and therefore without any side-effect introduced by other concurrent processes.

- **CPU Temperature:** We also tested whether significant environmental temperature changes would invalidate the fingerprint, as previous works have observed that the frequency of the quartz crystal increases with temperature [207]. During our normal experiments, the regular CPU temperatures were generally around 38 degrees Celsius. Hence, we tried to stress the CPU for 20 minutes at 100% load, successfully doubling the internal temperature (as reported by the internal sensor). However, even if under these conditions the clock skew reported in previous studies [161] should have resulted in a measurable difference in our timing experiments, we did not observe any variations or errors in our fingerprint identification. A possible explanation for this discrepancy is that, while the increase in temperature can impact clock-based measurements, our approach relies on the difference of two clocks physically located in the same machine. Therefore, both are likely impacted by the temperature change, thus reducing the effect of the higher temperature and compensating the changes introduced in their frequency. As a result, while the difference introduced by the temperature in one single clock may be relevant, the difference

in the delta between two closely-located clocks may be too small to affect our fingerprint.

- **Long-term Stability:** We evaluated if the generated fingerprint remains stable over time during a normal use of the machine. In this case, we repeated our tests respectively one and two months after the fingerprint was first generated and found no problem in the identification process.

We selected fingerprinting functions that can be executed without interruption on a CPU. This guarantee that the collected timing information is not affected by side-effects introduced by other concurrent processes, making the measure independent from the CPU and/or I/O workload of the machine. When running the native measurement, we checked it was executed without interruption by using transactional memory. However, we could not guarantee the same property when the fingerprint is executed remotely over the web. Therefore, the scheduler might have interrupted some of the executions, but this is mitigated by the multiple calls to the function performed in the fingerprinting generation phase.

8.2.4 CryptoFP

Since this clock-based fingerprinting method works with virtually any simple function, we selected one based on its general availability and on the possibility to generalize our results and compare our host-based and web-based approaches.

According to these criteria, the selected function should be available in different forms but in all possible system. In fact, since one of our goals is to implement a web version of this device fingerprinting technique, it should be available also in JavaScript, called by a wrapper in this scripting language.

Based on the results of our preliminary tests, we decided to implement our prototype by timing the execution of the pseudo-random generator APIs (e.g., `CryptGenRandom`/`RtlGenRandom` in Microsoft Windows). These cryptographic functions are available in every system and also are accessible through JavaScript, which meet all our requirements.

8.3 Host-Based Fingerprinting of Identical Targets

Since the common evaluation procedure used to measure fingerprinting techniques does not take into account several important features, we first propose our own methodology (detailed in §8.3.1) that is able to capture the two main omissions of previous approaches: (i) the impact of targets heterogeneity and (ii) the actual stability of a fingerprint within the same machine.

To evaluate CryptoFP, we implemented a native version of the algorithm. This version calls directly the function that generates a series of random numbers. We also repeated the tests described in §8.2.3.4, confirming that there was not effect introduced by the CPU load, internal temperature, or long-term stability of the fingerprint. We also conducted several experiments with a subset of different computers in order to properly tune the similarity threshold used by our algorithm, resulting in a value of 0.5 (i.e., two fingerprints are considered to belong to the same machine if there is at least 50% of positive matches when comparing them, as shown in Figure 8.2).

8.3.1 Methodology

The current evaluation methodology for fingerprinting techniques measures two features: the entropy of the fingerprinting and the size of the anonymity sets [162]. These are often used to replace other widely accepted and more conventional metrics, such as precision and recall, that are rarely used in this specific area as they provide a less precise image of the discrimination power of a fingerprinting technique. Therefore, we also decided to use similar measurements to be able to compare our results with those obtained in previous studies. In fact, since the fingerprint generation process in all major techniques results in a hash or in an identifier, it is possible to compute the entropy — i.e., a representation of global uniqueness — among a set of tested devices. Moreover, due to the nature of these methods, if a particular machine A has the same fingerprint of B and B matches a third machine C, C will always match with both A and B. This transitivity allows the computation of anonymity sets.

However, CryptoFP works differently and does not generate a unique identifier. Instead, it produces fingerprint information that needs to be compared with the one collected on other machines to identify possible matches. In other words, it produces some sort of fuzzy hash, which can-

not be simply matched against other candidates, but requires a comparison routine to compute the similarity among two values. Also, in our case, the final result is not a direct comparison of identifiers but a similarity score based on the described matching procedure. This approach has been intentionally designed to be more resilient to noise in the timing of the generation of random numbers and results in a greater accuracy. However, due to this design, the transitivity property does not hold anymore — thus making CryptoFP difficult to evaluate using entropy or anonymity sets as the obtained results (e.g., the entropy of our time matrix) would not be comparable with the entropy values of previous approaches. In our evaluatuon, we will use an adaptation of the anonymity sets.

8.3.1.1 **Homogeneous Scenario**

Previous experiments were performed asking users to visit a website hosting the fingerprinting code. Therefore, users were likely using a browser running on commodity computers with different hardware, software, and configurations. While this is a realistic experiment (we will also use the same to further evaluate the web version in §8.4.2), it fails to capture the discrimination capability of the method, as the check strongly depends on the heterogeneity of the tested machines. For instance, if there are no computers with the same specific set of characteristics in the dataset, a simple hardware test can differentiate each client with 100% certainty. However, both companies and universities often rely on large numbers of identical machines, which can greatly complicate fingerprinting. To take this into account, we propose a homogeneous scenario evaluation that includes the next points:

- **Homogeneity:** In order to provide homogeneity and test our fingerprinting technique with the same hardware computers rather than different computers, we performed our experiments using two groups of machines with perfectly identical software (installed through a disk image) and hardware components. The groups included 176 and 89 computers, respectively. Thanks to this setup we can identify whether our fingerprinting algorithm is really distinguishing *hardware imperfections* and to what extent it is possible to discriminate exactly identical hardware.

- **Stability:** We define the stability of the fingerprint as the ability to identify the same computer repeatedly. This measure has not been

tested before in many previous studies, as authors assumed the property to be true by default. However, there may be some circumstances, such as specific hardware availability, general CPU workload, and number of concurrent process, that can affect and jeopardize the identification. Therefore, we repeated the CryptoFP generation phase three times for each computer. Each measurement was performed ten minutes apart. We then compared all results to check if the extracted fingerprints were always matching.

- **Discrimination:** Since our fingerprinting does not produce a hash but it needs a comparison phase, we cannot use the common measures like entropy or anonymity sets. Instead, we adapted the anonymity set measurement to an *identical comparison set size* that translates the idea behind anonymity sets to the comparisons performed by our method. In this way, the size is no longer the number of computers with the same fingerprint, but the number of computers with the same number of positive matches with other computers. To make it more clear, we are presenting a simple example. Imagine four different machines: A, B, C and D.

  - A matches B
  - B matches A and D
  - C matches D
  - D matches B and C

  In this case A, and C have a set size of one, and B, and D set size of two (because they match two other machines).

We run our CryptoFP native implementation in the two different sets (commodity computers running Microsoft Windows 7) and measured the properties introduced above. Using the threshold empirically computed in §8.2.3.3 ($n = 1000$ and $m = 50$) the test took just a few milliseconds, although obviously the exact computing time depends on the specific machine.

8.3.2 Results

As described above, we present our results using the *Identical Comparison Sets* metric, which is an adaption of the well-known anonymity set method

for fingerprint evaluation, obtained by substituting "identical fingerprints" by "identical fingerprint comparisons". Therefore, in our particular cases we have a 0–175 possible values for identical comparisons for the first set of computers and 0–88 in the other, where 0 means that the particular computer had no match and the maximum value meaning the computer matched every other machine in the group.

Furthermore, we tested the stability of our method repeating the generation of the fingerprinting three times in each computer and validated that, in all cases from both scenarios, CryptoFP was always able to identify the computer. Regarding the discrimination capabilities, the native version of CryptoFP with a similarity threshold of 50% was able to distinguish every computer in each group. In other words, the uniqueness of our method in both tests is 100%, even when both hardware and software in the computers are identical. This shows that CryptoFP is capable of detecting clock crystal imperfections in order to accurately distinguish machines.

Please note that even thought we did not observe any in our experiments, collisions may occur on larger sets of identical targets. However, in most of the possible use cases, this is an acceptable result. In fact, if a user has a license bound to some machine, it is not very likely that she can test the software on tens of thousands of other identical machines just to find another one in which the software can be used. Our algorithm had no collisions in a lab containing 176 identical machines and another with 89 identical machines, which is a sufficient guarantee in most use cases.

8.4 Web Implementation of CryptoFP

The HTML5 Web Cryptography API is able to interact with cryptographic keys and functions managed by users. A very important aspect for our hardware-level device fingerprinting to work at native level even from the web is that "*the API itself is agnostic of the underlying implementation of key storage*" [279]. Its main objective is to provide just an interface or wrapper that allows system-level cryptographic operations such as hashing, encryption, or decryption.

This API offers several interfaces to cryptographic functions through the `window.crypto` or `window.crypto.subtle` properties. The implemented methods can be very simple such as `getRan- domValues` to generate a set of random numbers, `digest` to generate hashes, or `generateKey` that generates keys for encryption.

```
1  void RandBytes(void* output, size_t output_length) {
2    char* output_ptr = static_cast<char*>(output);
3    while (output_length > 0) {
4      const ULONG output_bytes_this_pass = static_cast<ULONG>(std::min(
5      output_length,
           static_cast<size_t>(std::numeric_limits<ULONG>::max())));
6      const bool success =
7      RtlGenRandom(output_ptr, output_bytes_this_pass) != FALSE;
8      CHECK(success);
9      output_length -= output_bytes_this_pass;
10     output_ptr += output_bytes_this_pass;
11   }
12 }
```

Figure 8.3: Extract from the Chrome Implementation of `generateRandomNumbers`.

```
1  size_t RNG_SystemRNG(void *dest, size_t maxLen)
2  {
3    size_t bytes = 0;
4    if (RtlGenRandom(dest, maxLen)) {
5     bytes = maxLen;
6    }
7    return bytes;
8  }
```

Figure 8.4: Extract from the Firefox Implementation of `generateRandomNumbers`.

8.4.1 Implementation

We selected the simplest method available in the API, namely `getRandomValues`, for our device fingerprinting technique. Since our method is a timing side-channel attack, a complex cryptographic method — although the actual operations are performed at native level — may obscure our timing and make our fingerprint dependent not only in the underlying cryptographic functions, but also in the Web Cryptography API itself.

We analyzed the implementations of this method in two major open-source browsers, Firefox and Chrome, and inspected the native cryptographic function calls which were performed when the function was invoked. For example, when running Microsoft Windows, in both Chrome

and Firefox , the `generateRandomNumbers` call finally leads to the native function `RtlGenRandom` to generate random numbers. For our experiments it is important, as shown in Figure 8.3 and Figure 8.4, that the browser API is just a basic wrapper for the native version, so the browser will not make other operations or memory accesses that may pollute the time measurement.

Regarding the values for n and m, we will use the empirically computed values of 1000 and 50 as indicated in §8.2.3.3. The computing time needed for the generation and checking of the fingerprint is just a few milliseconds. In order to determine the specific threshold for the web implementation of CryptoFP, we performed various preliminary tests. As the timing precision offered by HTML5 is smaller than the native timing functions, the threshold was finally set to 100% for the comparison of time matrix.

8.4.2 **Evaluation**

In this case, we compare CryptoFP with the other three state-of-the-art web hardware-level device fingerprinting techniques: (i) the famous canvas fingerprinting [199], (ii) the improved version of WebGL fingerprinting [48], and (iii) the recently discovered audio fingerprinting [83]. This allows us to compare the discrimination capability and stability of the four different techniques.

As the web implementation is devoted to track users on the Internet, we analyzed the fingerprinting techniques both in the homogeneous scenario presented in §8.3 and by using a classical web evaluation where users were asked to visit a website that performed all the techniques (making a total of 565 different users). In this case, we informed the users about our experiments, and ask permission to collect the information that was going to be gathered by our tool. Users where using their own machines and had no restriction on what computer they were using, so therefore our dataset can contain both GNU/Linux and Microsoft Windows in many different versions. In addition, in order to protect the users privacy, all the data collected was anonymous. We disseminated the URL of the website through social networks and friends, asking them to participate in the study and further re-disseminate the link among their contacts.

As described in §8.3, all results are shown using the *Identical Comparison Sets* metric, that is an adaptation of the extensively used anonymity set technique to evaluate fingerprinting methods. Zero indicates that there is not other match in the dataset, and the maximum number indicates that

the fingerprint is the same in all the computers.

8.4.2.1 **Homogeneous Scenario**

In our experiments, we tested the stability of our technique by repeating the fingerprint generation three times in each computer. We found that all methods correctly generate the same fingerprint in all our tests, with the exception of audio fingerprinting, that failed the stability test in 21% of the cases, thus raising serious doubts about its possible use as fingerprinting technique with a basic hash comparison, regardless of other factors. For this reason, audio fingerprinting was removed from the following discrimination capability tests.

All methods took just few milliseconds to execute, with the exception of WebGL that required several seconds. Regarding the possible overhead, all methods are simple enough to result in no observable slowdown, with again the exception of WebGL, which relies on complex graphics checks and can therefore slow down navigation while it is being executed.

We divided the comparison sets in five groups, one containing computers that did not share any fingerprint, then three equally divided groups containing respectively 1-58, 59-117, and 118-174 positive matches in the 176 computer group and 1-28, 29-57, and 58-87 positive matches in the 89 computers set, and finally, one group with computers that shared their fingerprint with all the rest. CryptoFP was able to cover around 18% of the computers with the two first sets for each of the computer groups (0-58 and 0-28 matches) and the percentage increases until 85% if we include the third set (0-117 and 0-57 matches). Even if these results are far from the perfect identification capability provided by the native method, current top state-of-the-art hardware-level fingerprint methods (canvas fingerprinting and the improved version of WebGL fingerprinting) could not differentiate any of the computers in none of the two homogeneous groups, resulting in the same fingerprint for all computers. Therefore, our solution clearly outperforms all previous state-of-the-art hardware technique in this particular settings.

Finally, the result of this experiment show that the web implementation of our technique is less precise than the native implementation, due to a more coarse-grain precision offered by the HTML5's `performance.now` timer. We will discuss different solutions in order to improve the results of the web implementation in §9.5. However, it is important to note that despite this limitation, CryptoFP is still capable of distinguishing completely

identical hardware and software computers.

8.4.2.2 Heterogeneous Scenario

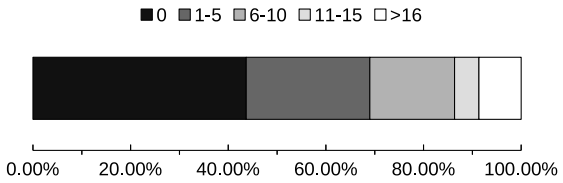
In this case, we also divided the comparison sets in five groups, but instead of separating the sets equally, we divided the sets every 5 matches, starting from 0 up to 15. The first group means that no additional matches were detected apart from its own, the second group counts the number of computer with 1-5 matches in the dataset of 300 computers, the next groups between 6-10 and 11-15 matches, and the last group counts the computers with more than 16 matches. In contrast to the homogeneous analysis, in this scenario, all the fingerprinting techniques are able to differentiate computers, so this more fine-grained set sizes will allow us to compare the methods more precisely.

Looking at the results collected thought our website, reported in Figure 8.5, we can see that CanvasFP obtains only around 10% of completely unique fingerprints and the improved WebGl FP around 15%, whereas CryptoFP achieves around 45% in exactly the same dataset. More in detail, CryptoFP covers 70% of all the involved computers with just the two first identical comparison sets (0-5). Specifically, more than half of the computer were either completely unique or only matched another computer. However, both CanvasFP and improved WebGl FP obtain only around 40% with the two first identical comparison sets, which is less than just the first set, unique fingerprints, of CryptoFP.

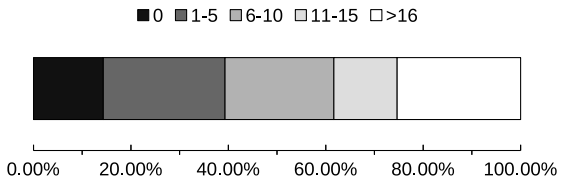
Obtained results show the capabilities of the web version of CryptoFP, which is outperforming all existing hardware device fingerprinting solution, being able to obtain a better discrimination also in a heterogeneous scenario.

Fingerprinting combinations CryptoFP, as any other device fingerprinting techniques, does not necessary need to work as a standalone solution. Instead, it can be easily combined with other different techniques, as other approaches already proposed to date. As a case study, we decided to combine all the hardware-level device fingerprinting methods with ours in order to increment the size of the discrimination rate by cross-referencing the results of the different methods.

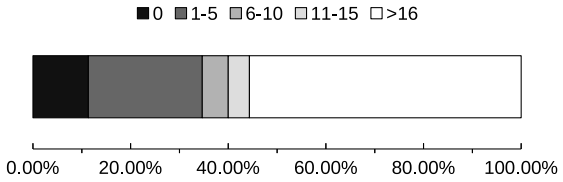
Figure 8.6 shows that the combination of the hardware-level device fingerprinting techniques (the stable ones) achieved a uniqueness of around 80% and nearly a 100% coverage by just including the second comparison



(a) Identical Comparison Set Sizes for CryptoFP.



(b) Identical Comparison Set Sizes for the improved WebGl FP.



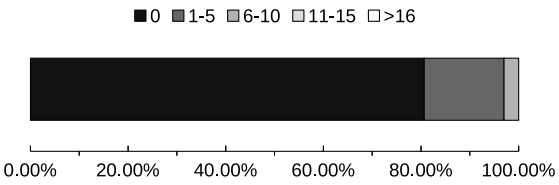
(c) Identical Comparison Set Sizes for CanvasFP.

Figure 8.5: Identical Comparison Set Sizes for CryptoFP, improved WebGl FP and CanvasFP in-the-wild web evaluation (300 different users involved). The colors represents the number of identical comparisons whereas the X axis represents the percentage of computers in the ranges.

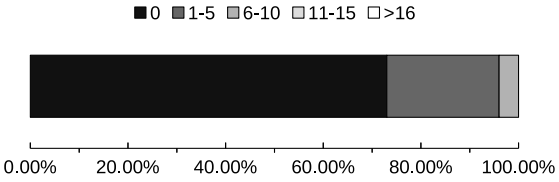
set (1-5). This simple combinations of CryptoFP with the improved WebGL FP and CanvasFP follow a similar fashion, with a 70% and 60% of uniqueness and nearly 100% and 90% coverage when the second comparison set is included.

8.5 Discussion

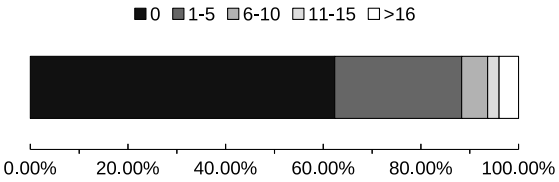
Generality The assumption behind our approach is that any function can be timed and that this timing information can then be used to fingerprint



(a) Identical Comparison Set Sizes combining CryptoFP, the improved WebGl FP and CanvasFP.



(b) Identical Comparison Set Sizes combining CryptoFP and the improved WebGl FP.



(c) Identical Comparison Set Sizes combining CryptoFP and CanvasFP.

Figure 8.6: Identical Comparison Set Sizes for the different combinations of CryptoFP with the rest stable hardware-level device fingerprinting techniques (300 different users involved). The colors represents the number of identical comparisons whereas the X axis represents the percentage of computers in the ranges.

subtle clock differences in the underlying machine. To confirm this hypothesis, we tested several functions in order to find out how generic the function selection can really be. After these preliminary tests involving functions of different nature, we realized that our method needs the function to be uninterrupted by the OS scheduler because, otherwise, the timing values would obviously be polluted by other processes. We also found that the timing of very small functions is also harder to measure, requir-

ing a higher number of iterations to obtain a stable value. Therefore, we can conclude that our method require a function that includes a sufficient number of instructions, but not long enough to be often interrupted by the scheduler.

The confirmed generic nature of our approach makes it adaptable to different environments and situations. For instance, if a certain installation of a particular operating system uses a restricted version of the standard C library, our method can easily be changed to use another installed function. Similarly, if the target uses a completely different version of the operating system, even dedicated to IoT systems or critical infrastructures, if we can learn which functions are available, we can easily adapt our method in order to work under this new environment.

If we can execute native code, we can also create our own function and perform the timing using this function — making our code completely independent from the system libraries, as long as we have access to a timing operation that does not use the CPU clock signal.

Fingerprint Evaluation In §8.1, we introduced a set of features that we hope can serve as guidelines for future fingerprinting evaluation. In addition, instead of testing our method against random machines, our evaluation procedure (described in §8.3 and §8.4.2) was designed to stress the algorithm in a scenario in which all machines have identical software and hardware components.

Table 8.2 summarizes the characteristics of different device fingerprinting techniques proposed to date, and compare them with our approach. Our method was the only one to discriminate all the computers (in the machine version) and the many of them (in the web version). In fact, the other methods could not differentiate any of the computers in any of the two sets. Stability was 100% for all methods, except of the Audio FP technique that returned different fingerprint values on the same computer. In addition, our method was the only one resilient to both changes and evasion techniques. In fact, since the method does not necessarily rely on a specific function, the only reliable way to affect its measurement is to insert noise in the time measurement — something that can have serious side effects on many web pages. Similarly, our fingerprint can survive even a complete re-installation of the operating system.

The only negative aspect of our solution, if used as a way to track users on the Web, is the back-end efficiency. On the one hand, computing a single fingerprint is extremely fast. On the other hand, existing fingerprints

Table 8.2: A comparison of current state-of-the-art methods according to the proposed features. ✓ indicates that the method has, to a certain extent, that characteristic. ✗ implies that either the method has been tested and does not meet the feature or that, because of its design, it is unlikely to meet that requirement.

	Methods				
Feature	**Attrib.-based FP**	**Canvas FP**	**WebGL FP**	**Audio FP**	**Our method**
Discrimination Power	✓	✓	✓	✓	✓
Stability	✓	✓	✓	✗	✓
Homogeneous Discrimination	✗	✗	✗	✗	✓
Efficiency	✓	✓	✗	✓	✓
Resilience to Evasion	✗	✗	✗	✗	✓
Resilience to Changes	✗	✗	✗	✗	✓

cannot be just indexed in a database for a fast retrieval. Instead, our solution require to compare a new fingerprint with all those collected for other machines. However, each comparison is fast (200 milliseconds in our current Python prototype), completely independent, and easily parallelizable. Moreover, an incremental comparison can be implemented to optimize the process, stopping the algorithm and removing candidates when a difference is found.

Application to Web Device Fingerprinting The web-based implementation of our algorithm was not as precise at discriminating identical hardware and software machines as the native implementation. The reason behind this fact is the granularity of the HTML5 timing API, which does not allow for a more precise measurement. However, there are several improvements that can be implemented in the web version to enhance the timing precision.

First of all, instead of using the standard HTML5 timing API, there are improved timing techniques that can achieve more precise timing values, such as the clock interpolation technique presented by Schwarz et al. [248]. The timing precision we can obtain with some of this timers is similar to the timer used in the machine version. Therefore, it is logical to think that the fingerprint should also be as precise. Even in this particular case, the evasion would be difficult to implement since the functions used can be easily modified.

In addition, *WebAssembly* [124], a project that aims at introducing a new binary format for web applications, can also be used. In this case, we may not only improve the precision of the web version of CryptoFP but also im-

plement a web version using any function. This API will allow to compile C/C++ code, amid others, as well as execute it at native speed using common hardware capabilities. The technology is currently in an early stage but it can be used in the future to fully implement the native fingerprinting method.

Countermeasures Regarding possible evasions, we did not test those in which users were performing specific actions to tamper with the results – such as underclocking/overclocking the CPU, but we focus instead on techniques implemented by browsers to avoid fingerprinting. In fact, some of the existing fingerprints are ineffective against existing browsers countermeasures. As our technique does not necessarily rely on a specific function, such protection is more difficult to implement.

Nevertheless, there are few countermeasures that can be adopted in order to avoid our new fingerprinting method. Since the basis of our method is the precision of the timing process itself, countermeasures need to focus on this aspect. While this is possible in the context of a browser, major browsers have already reduced the precision of their timers to avoid several of these attacks performed by JavaScript. Reducing it even further would definitely be an unpopular solution, as more and more applications are pushing for better timing capabilities in JavaScript and HTML5.

Another countermeasure could rely on the use of secure timers, several of which have been proposed in the literature [182, 261, 153]. Their goal is precisely to control timers to make attacks more difficult. These methods are, nevertheless, costly to implement [122].

8.6 Related Work

Physical Device Fingerprinting Physical device fingerprinting relies on variations in physical features of devices for their identification. Originally intended for authentication, other uses appeared over the years, such as license binding or statistically determining the source of an attack [96]. Another work focused on wireless device fingerprinting [30, 107] tries to identify a network source rather than a machine. Other techniques have been proposed to physically identify hardware. Examples include the variation in the process in semiconductor foundries [37, 39, 212], Physical Unclonable Functions (PUFs) [223, 112, 165], and exploiting motion sensors embedded on smart devices [71, 64].

Another line of work [227] focused on fingerprinting computers based on the system clock skew extracted by analyzing the different types of timestamps present in the generated traffic. Kohno et al. [154] exploited the TCP and ICPM timestamps to identify computers. Later, Jana and Kasera [138] used the timestamp present on WLAN beacon packets to identify unauthorized wireless access points. More recently, Huang et al. [134] proposed to use the Bluetooth included in some devices to identify the skews. These techniques are really interesting, but the information they rely upon are optional and not always enabled by default in various operating systems and can be easily spoofed by the user. Moreover, they can be easily disabled by users, thus completely preventing the fingerprint computation. Our approach follows instead a schema that allows to obtain a fingerprint without relying on any specific options in the system and without needing to analyze any traffic data, and still allowing a precise identification of computers, even if they share the same hardware and software.

The works closest to our is the proposal to use Flash memory to produce both random numbers and generate unique device fingerprints [281] and the proposal to use a clock crystal fingerprinting technique that by using another time reference [237]. However, these approaches differ from ours, because ours only relies on timing functions to fingerprint hardware, being less dependent on the specific hardware configurations. In addition, we have been able to create a generic and simple version of clock fingerprinting that can be used both in simple native code and in the web environment.

Browser Timing Attacks Timing attacks were first introduced by Felten and Schneider [92] to acquire users' information. Bortz et al. [38] categorized timing attacks into two different categories. The first attacks consisted in measuring the time differences through direct timing. The second ones use information from different sites to obtain client-side data.

The usage of CSS properties can also be a source for timing attacks [155]. Van Goethem et al. [276] proposed the usage of the size of cross-origin resources to detect previous access. Sanchez-Rola et al. [244] discovered installed extensions in all major browsers based on access control settings by means of a timing attack. Mowery et al. [198] presented a method using JavaScript engine benchmarks.

Web Fingerprinting Web fingerprinting is a method to retrieve user or browser information, typically for tracking. *Cookies* [257] were their first

form. Later, it started to be more complex e.g., *evercookies* [147], *cookie syncing*, or *ETags* [26]. Finally, *device fingerprinting* computes a unique identifier for each machine without client-side storage.

There are two types of device fingerprinting: *attribute-based* and the most advanced*hardware-level*. The first one uses browser attributes [79] (e.g.,installed fonts or plugins, `UserAgent`, or screen size and resolution). Unfortunately, these attributes change rapidly, rendering the fingerprint obsolete in less than 10 days according to [278]. The second one, however, uses browser implementations of different APIs to compute the differences between devices that are based in hardware features (e.g., HTML5 Canvas API or the WebGL API [199]).

8.7 Conclusions

Device fingerprinting is an active research topic within web security, specially web device fingerprinting, in the last years. These methods can be used for a wide variety of tasks such as user access control, web tracking or analytics, or targeted attacks.

In this chapter, we introduced a time-based device fingerprinting technique. This fingerprinting technique is generic and can work with different functions, making the method adaptable to different environments. In addition, we introduced a set of properties to properly assess the functionality of fingerprinting techniques, filling the gap in current fingerprinting evaluation and proposing a new homogeneous scenario evaluation procedure.

We built a specific native version of our method, CryptoFP, using the function for generating random numbers and evaluating it in a homogeneous scenario with two large sets of machines with the exact same hardware and software installed, showing that is capable of distinguishing every machine. Based upon this implementation, we built an application to web device fingerprinting using the HTML5 Cryptography API that internally uses the same native functions , evaluating and comparing it with state-of-the-art hardware-level fingerprinting . In a homogeneous scenario evaluation CryptoFP was not as accurate as its native counterpart due to the timing limitations of the JavaScript engine, but still capable of discriminating several of the identical machines, outperforming the state-of-the-art methods that were not able to uniquely identity none. The heterogeneous evaluation shows that the percentage of unique computers identified by CryptoFP was much higher than any other existing method.

"My hope still is to leave the world a bit better than when I got here."

Jim Henson (1936–1990)

Chapter 9

BakingTimer: Privacy Analysis of Server-Side Request Processing Time

The World Wide Web (WWW) is built on top of the HyperText Transfer Protocol (HTTP), a stateless request/response protocol in which each request is executed independently from any other received before. However, most web applications need to keep track of the user's progress and status from one page to another. For instance, after a user has successfully logged into a service, she expects the application to remember her authentication and allow her to perform other actions in the same website accordingly.

In order to solve this problem, in 1994 Netscape Mosaic introduced the use of Cookies [265]. HTTP cookies are small fragments of data that servers can send to the users inside their responses (by using the `Set-Cookie` header field) or by the use of JavaScript code executed in a webpage in order create a cookie on the client-side (by invoking the `document.cookie` function). Either way, the browser stores the value of each cookie and includes them to every future requests made to the same server. Today, cookies are used for a variety of different purposes, including to maintain the users' login status, to store different options that a user makes while browsing a websites (such as the language preference or the accep-

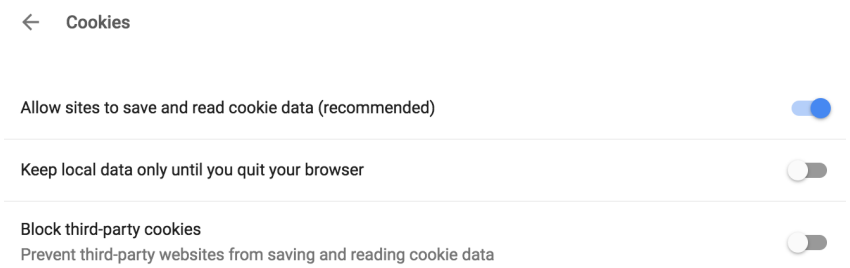


Figure 9.1: Chrome Cookie Settings.

tance/rejection of a specific consent), or to simply keep track of previous visits from the same user.

The cookies we described so far are called *first-party* cookies, as they are created by the website the user is visiting. However, these are not the only cookies that may be created in the browser. Websites load different kind of resources to offer their services and the requests made to retrieve these resources may also create cookies. In many cases, these *third-party* cookies are used to track users around all the different websites they visit. For instance, several studies have measured that a big percentage of websites on the Internet perform some form of user tracking [83, 215, 14, 243].

While tracking based on third-party cookies is one of the main techniques different companies use to offer personalized advertisement, other alternatives exist — for instance based on fingerprinting the browser or the user's machine by using different hardware or software information [244, 48, 199, 162]. These approaches can completely bypass all the browser protections regarding basic tracking but their fingerprinting code needs to be executed in all the websites the user visits. Therefore, for websites that are not part of the main tracking networks, there is another option based on the so-called *history sniffing*. History sniffing attacks can track the user without relying on any code executed in other third-party websites, just the one executed in the website accessed by the user. While many methods have been proposed in this line of research [92, 38, 155, 166, 276], most of them suffer from several severe restrictions. For instance, they only provide coarse-grained classification (such as logged in vs not logged in), can be easily defeated by users without serious side-effects (such as by deleting the browser cache), and are all tested on a handful of anecdotal cases. As an example, the two most recent works in this area, published in 2015 at the ACM CCS [276] and NDSS [166] conferences, were only evaluated on, respectively, five and eight target websites.

In this chapter, we present a new method called BakingTimer that exclusively uses cookies of third-party websites. We are able to obtain part or the whole browsing history of a user simply executing some small piece of code in JavaScript. This code does no need any special rights and can be executed from any script loaded in the website, first or third party.

The remainder of this chapter is organized as follows. §9.1 provides the reader with the required background to understand the contribution. §9.2 details `BakingTimer`, our proposed attack that abused the current server request processing schema. §9.3 describes the analysis of the current privacy issues in several websites in order to measure the actual impact of this attack and propose countermeasures to be adopted against it. §9.4 describes several case studies extracted from the large-scale analysis that provide interesting insights about particular web scenarios. §9.5 discusses the main contributions of this work. §9.6 presents the related work performed in this area and contextualizes our work. Finally, §9.7 concludes the chapter.

9.1 Background

9.1.1 Browser Cookies

Cookies were introduced by Lou Montulli [247] while working in Netscape, and are considered the first method to track users on the Web. Cookies allow services to remember a particular user by storing snippets of data on the users' computers, maintaining a concrete browsing state for a returning visitor. After cookies were first presented, they were rapidly embraced by the web community because of their flexibility and the increased usability they enabled in a broad range of websites. As a result, they are now playing a core role as part of the Internet technology [215].

While cookies were not specifically designed to track users across websites, the abuse of their stateful nature started shortly after their first appearance. Since websites are composed of different resources that may be stored in the same domain hosting the page as well as in other third-party servers, these external resource providers have the capability of including cookies along with their provided resource. Therefore, if a third-party server provides resources to a large number of websites, it can certainly gather an accurate picture of a user's browsing history and profile her habits. For example, a website `site.com` includes an image from a third-party domain called `advertisement.com`. The server `advertisement.com` sends the image along with a `HTTP Set-Cookie` header, that will be stored

on her machine. When the same user visits another website `site-two.com` that also utilizes images from `advertisement.com`, her browser will send the previously set cookie to `advertisement.com` alongside the request for the image. This allows the advertisement server to recognize the user and collect a list of the websites she regularly visits. This behavior, called *third-party cookies*, is arguably the most widespread technique for web tracking.

However, even the most privacy-invasive third-party cookies are an important part of the web. In fact, the most common usage of this specific type of cookies is web advertisement or analytics. Their web tracking capability is used to track users' browsing history and use this information to build a user profile for custom advertisements. In fact, advertising companies have already stated that web tracking is required for the web economy [254]. However, other techniques exist that can be used for analytics and targeting while preserving users' privacy (e.g., [272, 126, 109, 34, 27, 18]).

However, both third-party and first-party cookies are widely used and browsers even encourage their usage in order to ease usability. For example, when a user of the well-known Chrome browser decides to erase all the stored cookies, the browser warns the user that "*This will sign out of most websites*". Even more important are the settings regarding cookies (shown in Figure 9.1). Chrome recommends to enable the permission for websites to read and write cookie data and permits to block third-party cookies. Therefore, the importance of cookies is not only acknowledged by websites, but also by the browsers themselves albeit the privacy issues that may arise.

9.1.2 History Sniffing

Even though it is not a common action among the average web users, *third-party cookies* can be blocked (as detailed in §9.1.1). However, the user cannot remove just these cookies, as browsers do not allow to specifically remove some of them but all the cookies stored. The only option would be to manually remove them one by one. A large variety of history sniffing methods exists. We can group these techniques in two categories: CSS-based and time-based.

CSS-based attacks. A common approach a page can use to detect if a user has visited other websites (out of a predefined list of targets) is to check the CSS style used to render links. For instance, by checking the CSS `:visited` style of a particular link [59] it is possible to tell if that link had been visited before (typically as it is displayed in a different color). Similarly, it is

possible to craft other CSS-based techniques [256, 130, 140, 288] even using filters to exploit the differences of the DOM trees [283] by using a time side-channel attack.

Timing-based attacks. There is a broad range of history-stealing techniques based on timing information. These approaches were first introduced for this purpose in order to compromise users' private data by measuring the time differences in accessing different third-party content [92], by discovering if it had been cached by the browser. Extracting users' true geolocalization for the same purpose is also possible through web timing, due to the high customization present in current websites. It is also possible to detect if the user is logged in certain websites, checking the time of specific request [38], exploiting the AppCache [166], or estimating the size of resources [276]. As explained in §9.2, our own attack belongs to the this timing-based attack category.

Countermeasures and Shortcomings Some of the attacks above are already mitigated by the browser, such as the CSS `:visited` style check. All the related browser functions (e.g., `getComputedStyle` and `querySelector`) will always return that the user has never visited the link [203]. Despite these mitigations, recent work has shown that the attack is still possible using new features available in modern browsers. However, several possible defenses exist to avoid the problem, such as the ones proposed by Smith et al. [256]. In fact, one of the techniques presented has already been solved in recent versions of Google Chrome [120]. On the contrary, we believe that server-side timing-based attacks as the one presented in this chapter are much more difficult to prevent, as we discuss in details in §9.5. Moreover, previous attacks could only identify one individual state (e.g., "accessed" or "logged") but were unable to distinguish among multiple different states with a single check. Finally, some of the previous attacks can even be easily mitigated by simple user actions. For all these reasons, we believe our method outperforms all previously-presented techniques for history sniffing.

9.1.3 Threat Model

In the timing attack presented in this chapter, we adopt the same threat model used by previous work in the area [38, 276]. In particular, we assume an attacker can run JavaScript code on the client browser to perform time cross-origin requests. This code can be loaded either directly by the first-

```php
<?php
$userID = "hKdfYw4y6mGFRrjKAcbAjc1M57dsTpKSQ";

if (isset($_COOKIE[["consent"]) {

    if (isset($_COOKIE[["userID"]) {

        if ($_POST["userID"] == $userID) {
            getUserData(); // Case C
        }

    } else {
        saveNavigation(); // Case B
    }

} else {
    askConsent(); // Case A
}
?>
```

Figure 9.2: Example code of a PHP server presenting the three different possible cases of a cookie process schema.

party website or by third-party services (e.g., advertisement and analytics companies).

This information collected by our technique allows an attacker to know which websites were previously visited by the user and on which websites the user is currently logged in. There are multiple usages for these data that can result on serious security and privacy implications. The most obvious use for the data is related to advertisement as the usage of the browsing history allows to display targeted advertisements. Moreover, an interested tracker could create a predefined list of websites and generate a temporal fingerprint of various users, indicating the user's state in each of them. Even if the fingerprint could not be used as an standalone fingerprinting solution, it will definitely improve the fingerprinting capabilities of other web tracking techniques. Finally, from a security point of view, this information can be used to perform selective attacks against particular victims.

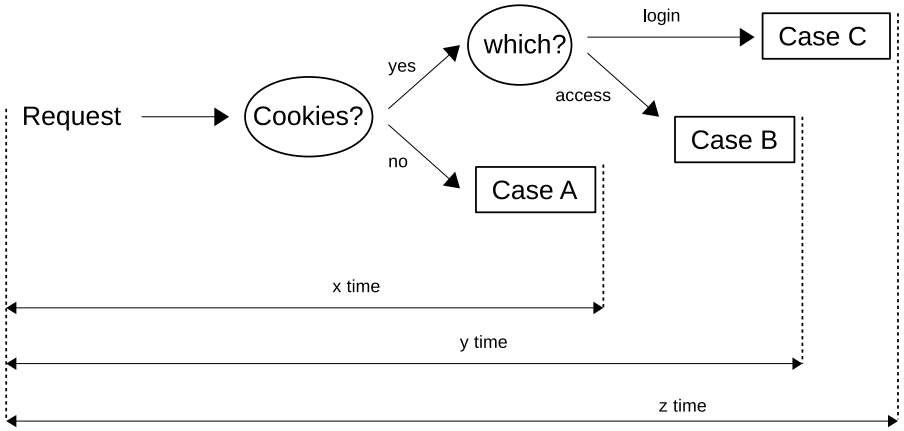


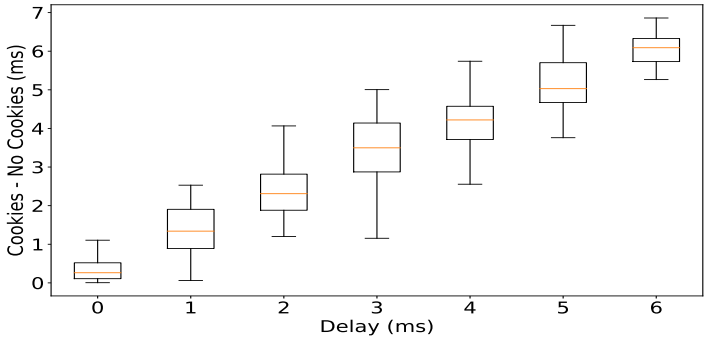
Figure 9.3: Server cookie procress schema.

9.2 BakingTimer

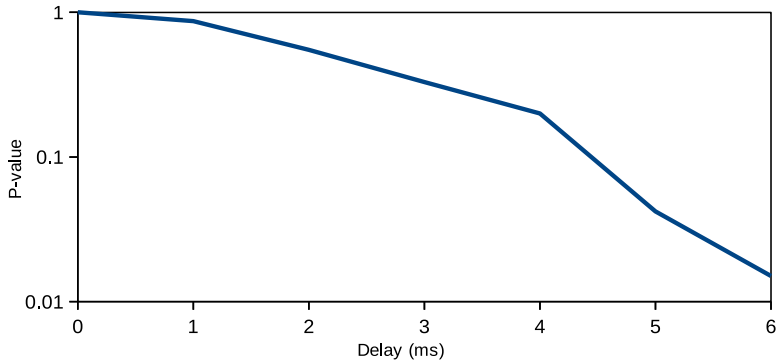
Out of all the requests a web server receive, some have cookies and some do not. If they do, it is reasonable to assume that the server-side application checks their value, may retrieve the associated session and looks for additional data from the database, or just ends up taking a different execution path.

For instance, Figure 9.2 shows a simple PHP skeleton that empathizes three different possible cases. First, the program checks if the user already accessed the website before by testing for the presence of the `consent` cookie using the `isset` function. If the cookie is not found (either because it is the first time the user access the website or because it has been deleted), the program takes the path we named *Case A* that calls the `askConsent` function. Otherwise, the program performs additional checks by looking for some login information stored in the cookies, *Case B* is followed that calls function `saveNavigation`, else, userID information is validated and then follows *Case C* with `getUserData` function.

Figure 9.3 shows a simplified representation of the control-flow of our toy application, emphasizing the different paths. Our hypothesis is that the time difference among the three cases is sufficient, even across a network connection, to tell the three behaviors apart and therefore to accurately identify whether cookies are submitted alongside an HTTP request. In particular, there are two main tests that make detecting a timing difference possible: the first to verify if there are cookies at all and the second to analyze them and load session data. While the comparison themselves are too fast to be identified, the different functions invoked in the three cases may



(a) Time deltas at different delays introduced in the server.



(b) P-value of the T-test at different delays introduced in the server.

Figure 9.4: BakingTimer resolution test.

not be. In our toy scenario, these results in the request take x seconds to be processed if no cookies are found, y seconds if they exist but there is not a current active session, and z seconds if the user is currently logged in.

Our prototype tool, called BakingTimer, performs cross-origin requests towards a number of target websites. Each request is performed multiple times, with and without cookies. In fact, while JavaScript code cannot directly test for the existence of cookies belonging to other domains, it can issue HTTP requests in two different ways: (i) by letting the browser sending cookies (if any exist for the target domain) or (ii) by forcing the browser to make a request without cookies. The time difference between the results is used by our tool to detect if the browser already had cookies for that particular target.

9.2.1 **Retrieval Phase**

The first phase of our approach is the retrieval phase, in which the algorithm computes the time required by servers to follow theoretically different paths, according to hypothetical cookies sent by the browser (see Figure 9.5 for the detailed pseudo-code of the algorithm).

To obtain the time information, we use the *User Timing API* [61] implemented in every major browser. The best solution to get a clear representation of the time, independent of specific temporal load or speed of the network, is to perform a comparison with two different times. Both time information are obtained side to side, so the workload of the network will likely be roughly the same in the two cases. To make it more clear, let us consider a simple example. Imagine we are timing the execution time of certain function of the server that directly depends on the existence of a specific cookie set on the user browser. Our system would execute a measurement by executing side-to-side the request with and without the cookies. In a first measurement, when the network workload it low, the system can obtain a value of 30.5ms when cookies are sent and a value of 20.3ms without cookies — thus estimating the execution time of the function with 10.2ms. In the second attempt, when maybe the network load is a higher than before, the same experiment can return two different values, such as 45.1ms with cookies and 34.7ms without — leading to a similar (even if not exactly the same) time estimation.

Following the schema presented in Figure 9.3, if a request requires x *seconds* to be processed (as opposed to y or z seconds), a simple comparison can be used to calculate the actual state of the user in relation with the website.

The first request, the one we named as *Case A*, is issued without cookies by using:

```
xmlHttpRequest.withCredentials = false;
```

This guarantees that the server would not receive cookies, even if the browser would have normally sent some for that particular website. Then, our code repeat the request, this time by setting

```
xmlHttpRequest.withCredentials = true;.
```

It is important to remark that the cookies sent in this second request (if any) are completely invisible for the website if it is not from the same domain. Even if the request includes ten different cookies from that domain,

the JavaScript code is not able to read that information. However, if we time the request we can obtain information that can later be used to classify the request. If the cookies are just cookies created in a simple access, the server will fall in *Case B* and if some login cookies are in the browser the time would be more consistent with *Case C*.

To make it more clear, we present a simplified example.

- **Not Accessed:** If the user never accessed the website, and we perform the timings described above, we will obtain a result close to zero. In fact, the first request will take x seconds, because we made the request without cookies. However, also the second request with cookies will take roughly the same amount of time, as no cookie were available in the browser for the target. In this situation we can infer than both calls were the same, indicating no previous access to the website under inspection.

- **Accessed/Logged:** If the user accessed the website in the past or if it is currently logged in into it, the result of the comparison of the time taken by the two request will be different from zero. As we are not using absolute values, but only time differences between consecutive requests, our technique can compensate for differences due to different network delays or server workload.

  A more fine-grained classification to distinguish between previous access and current login is also possible if the attacker has information about the different time required for specific actions. We will investigate this option and provide more details in the comparison phase section.

The algorithm (see Figure 9.5) takes one parameter n that indicates the number of comparisons to perform. As servers can have specific changes in their workload for many different reasons independent of our tests, we decided to obtain more than one simple comparison. Clearly, the more comparisons the attacker is able (or willing) to perform, the more precise is the model of the server-side computation she can obtain. Nevertheless, there is an obvious trade-off between the time spent in fingerprinting a singe website and the number of websites the attacker wants to test. We will investigate different options and the impact of the number of comparisons on the accuracy of the classification in §9.3.

Input: n the number of comparisons to perform.
Output: cs an array of arrays of numbers representing the server request processing schema: each position are the result of timings with and without cookies.

```
Function bakingTimer (n)
    i ←1  cs ← float[][] of size n×2  while i ≤ n do
        j ← 1  while j ≤ 2 do
            startTime ← GetCurrentTime()  if j % 2 = 0 then
                Request(cookies)
            else
                Request()
            end
            endTime ← GetCurrentTime()  logTime ← endTime −
             startTime  cs[j][i] ← logTime  j ← j + 1
        end
        i ← i + 1
    end
    return cs
```

Figure 9.5: BakingTimer retrieval process.

9.2.2 Comparison Phase

As explained in the previous section, our code performs a number of measurements by timings different HTTP requests for a target website. By subtracting each pair of requests, with and without cookie, we obtain an array of n time deltas.

In order to classify this data, we need to obtain a ground truth dataset for the selected websites. To this end, we need to retrieve the computing time information for each of the cases we are interested to detect. By using this information, we can then statistically test whether a set of measurements belong to one group or another, simply by calculating a T-test on these two data samples. This is a two-sided test for the null hypothesis that two samples have identical average values. This result will tell to which of the three presented cases does a certain experiment correlates with.

9.2.3 BakingTimer's Resolution

Before we started our large scale experiments, we wanted to experimentally verify that our hypothesis is correct and that the time difference among dif-

ferent server functionalities can be successfully distinguished over a long-distance network connection. For this reason we implemented a toy service based on `web.py` [266]. a simple but powerful Python framework that has been used by famous websites such as Reddit and Yandex. In our example, we controlled the time spent in each path by invoking the function `sleep` from the `time` library. All HTTP requests sent to the service were issued from a browser located in a different country of were the server was located [1]. to avoid any possible bias introduced by short-distance connections. The average ping time among the two machines was 13.6 miliseconds.

The server-side application was designed to introduce no delay for requests without cookies, and a configurable delay (ranging from 1 to 6 milliseconds in our tests) to the processing of any request containing cookies.

Results of our experiments are summarized in Figure 9.4. The graphs show that it is possible to detect the time the server took in each request quite precisely (see Figure 9.4a), despite the network or server load. However, with delays below five milliseconds, the difference among the two requests measured by the browser is not statistically significant and therefore an attacker could not conclude whether or not cookies were present in the browser for the target website (see Figure 9.4b). Nevertheless, if the difference between two paths is equal or above five milliseconds, then, even over the network, it is possible to reliably tell the difference between the two cases. This indicates that it is not necessary for a website to perform complex computations to be vulnerable to our technique and even looking up some data from a database may be sufficient. Obviously, less optimized servers are more prone to incur into larger delays than are easier to observe remotely, while high-end servers may make the path detection more difficult and error prone. We will analyze all these situations in the following section.

9.3 Real-World Experiments

Most of the previous study on history sniffing limited the experiments to just some case studies to prove the validity of the presented approach. This poor coverage is mainly due to the fact that they could generally only distinguish between two states: currently login or not logged in. As a result, experiments required the authors to manually create accounts for different

[1] Real locations anonymized for submission

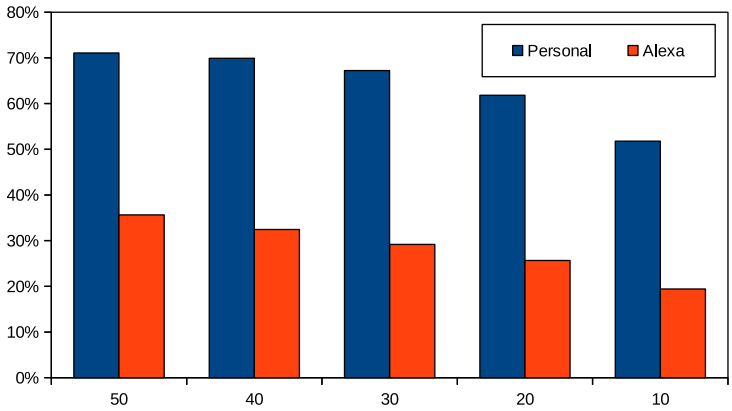


Figure 9.6: Percentage of vulnerable websites depending in the number of comparisons performed.

services, thus reducing the number of websites that could be tested.

Our technique allows instead to distinguish among multiple states, and in particular to differentiate between websites that have been previously visited by the victim from those that have not. Therefore, on top of performing a login detection case study (detailed in §9.4), we also conducted a large scale experiment to measure the percentage of different types of websites that are vulnerable to our attack.

The dataset used in our tests consists of two different groups: (i) highly accessed websites and (ii) websites related to sensitive information that users may want to keep private. For the first group we selected websites from the Alexa [252] Top5K list, to verify whether extremely popular services are also vulnerable to our time-based side-channel attack. The second group is composed instead by websites that can be used to detect private personal information of the user, such as medical or religious websites. Since many of the entries in this group are not included in the Alexa Top1M, so we can also check if not highly accessed websites are more or less vulnerable than highly accessed ones.

9.3.1 **Domain Selection**

We first populated our personal information website list using multiple queries obtained through the auto-complete option offered by search engines (e.g., *"cancer treatment side effects"*). We made five different queries for all of the next six categories: medical, legal, financial, sexual identity, political, and religion. Our tool issues each query on four search engines

(Google, Bing, Yandex, and DuckDuckGO) and retrieved the top 100 results from each of them.

To avoid an overlapping between the two lists, we removed from this group domains that also belonged to the Alexa Top10K list. We also removed any website that appeared in multiple categories, as we wanted to focus only on specific areas in isolation. Finally, we obtained a set of 5,243 unique personal information websites. In order to balance the two groups in the dataset, we selected the same number of websites from the Alexa top list. The combination of the two groups resulted in a final dataset of 10,486 websites.

9.3.2 Methodology

We implemented our BakingTimer proof-of-concept using a custom crawler based on the well-known web browser Chrome, that uses `xmlHttpRequest` for the requests. The crawler follows two different phases in order to obtain the data that we will later used to check if a target server is actually vulnerable to the attack.

- **Never Visited:** First, the tool cleans every previous access information stored in the browser. Then, it starts making the different requests described in §9.2 from a third-party website (blank tab). This data will be later used as a baseline for requests performed when no previous access was done.

- **Previously Visited** In this case, the browser first accesses the website under inspection. No cleaning process is performed, so all cookies automatically created by the website are saved in the browser and sent to the server in the following requests. After that, it goes to a third-party website (blank tab) and starts making the different requests described in §9.2 to that same website under inspection.

Once all the data was retrieved, we performed the statistical tests described in § 9.2 in order to identify whether the time information in the two groups of requests are statistically different or not. We also repeated the experiment with different number of requests in order to check their influence on the final result. To be practical, we tested with a minimum of 10 and a maximum of 50 comparisons (therefore ranging from 20 to 100 HTTP request per target website). The higher the number the more stable is the

Table 9.1: Percentage of website in the different categories that are vulnerable to our attack.

Category	% Vulnerable
Sports	43.75
Search/Portal	42.25
Government	40.63
Travel	40.40
Gaming	39.39
Adult/Pornography	39.06

measurement, but so is the amount of time required to run the test. Therefore, the choice boils down to a trade-off between precision and scalability. We believe that if the attacker is only interested in a handful of websites, then it would be possible to perform even more than 50 comparisons.

Out of the total 10,486 websites we analyzed, even with the minimum number of comparisons, nearly 40% were vulnerable to the BakingTimer attack and allowed third-party websites to know whether a user had previously visited them. The percentage of vulnerable websites increases to more than a half (53.34%) if the page performs 50 comparisons. Figure 9.6 shows the success rate at different numbers of comparison for both groups separately.

9.3.3 **Highly Popular Websites**

This group includes 5,243 websites from the Alexa top websites list. As websites in these categories are visited by a large number of users, most of their servers and the code they run are likely more optimized, thus making more difficult to measure the timing side channel. This is confirmed by the fact that our attack worked on 35.61% of the websites in this category. This result is still quite severe, as it means than a webpage could still reliably identify if its users had previously visited more than one third of the most popular pages on the Internet.

In order to get a deeper analysis of the obtained results, we clustered the websites in different categories and we computed the percentage of each of them that were vulnerable to our attacks. To determine the category, we used three services: Cloudacl [58], Blocksi [36], and Fortiguard [100]. Their category names are similar, and, after a normalization process, we settled for 78 different category names. Table 9.1 shows that the top 6 categories

Table 9.2: Sensitive website categories vulnerable to BakingTimer.

Category	% Vulnerable
Medical	72.63
Religion	71.66
Financial	71.63
Political	70.73
Sexual Identity	70.38
Legal	69.39

vulnerable to the attacks include around 40% of the websites, with a peak of 43.75% in the case of sport-related websites in the Alexa top list.

9.3.4 Privacy-Sensitive Websites

This group includes 5,243 websites from six different categories related to private personal information — i.e., medical, legal, financial, sexual identity, political, and religion. The results from these websites allows us to understand two different aspects. First, we can verify the amount of websites directly related to sensitive information that are vulnerable to our attack. Second, it gives us an opportunity to test less popular websites (as 85% of the vulnerable sites in this category are ranked below the Alexa Top500K) to observe whether smaller server infrastructures can result in a higher accuracy of our technique.

The results of our experiments show that a stunning 71.07% of all the analyzed websites in this group are vulnerable to the BakingTimer attack. If we break down the result by category we see that all have similar percentages and there is no clear difference between them (see Table 9.2). This result is much higher than the one obtained in the top Alexa group. And the difference is not due to the number of cookies. In fact, we compared the mean and standard deviation of the number of cookies in privacy-sensitive websites and highly popular websites. Unsurprisingly, the results show that highly accessed websites have a higher mean number of cookies (9.03±7.87) compared to the number of cookies in private personal information websites (5.83±5.49). This means that the main reason behind the difference is not linked to the number of cookies, but more likely to the slower servers or the less optimized code responsible to process the incoming requests.

9.4 Login Detection

In this section we look at performing a more fine-grained classification of the user history, not just by telling if a victim has visited a target website in the past, but also to distinguish a simple visit from owning an account or being currently logged in into the service.

Since this analysis requires a considerable manual effort to set up the accounts and operate the websites, we will limit our study to few examples taken from the two groups in our dataset. This approach is similar to the one regularly adopted by other related work on history stealing techniques [166, 276]. It is important to remark that we did no test websites that required pay accounts, lengthily registration procedures (e.g., requiring a phone number verification or a bank account).

9.4.1 Highly Accessed Websites

From this category we selected two popular websites, related to gaming and clothing, that were detected to be vulnerable to our BakingTimer attack (§9.3), and that have been the target of different phishing attacks — in one case for the huge number of users and in the other case for the high economic value of their customers. We are talking about World of Warcraft and Gucci.

In both cases, after the user logs in, a number of cookies are created in the user's browser to store this new status (e.g., `wow-session` and `gucci-cart` respectively). The presence of these cookies make the server take a different execution path, resulting in a different computation time. We followed the same analysis principles as the ones used when we analyzed the websites in §9.3, and detected both websites fall in our simplified three-paths model presented in Figure 9.3. The results show than each of the different states (e.g., not accessed, accessed, and logged), does not match any of the other two states when performing the statistical test of the comparison phase (see §9.2).

In this case, a third-party website can check if the user ever accessed any of these websites and can even check if the user is logged. This type of information could be extremely useful for a malicious attacker that is looking for a possible victim of a targeted phishing attack. Actually, the attacker does not need to control the third-party website, just have his code executed on them, for example via an advertisement.

9.4.2 Private Personal Information Websites

Detecting if a user has previously visited a website linked to specific information such as religion or sexual identity can leak some private information to third-parties the user may not even be aware of. However, if the third party could actually detect that the user is currently logged in into one of those websites, the link between the user and the website becomes much stronger.

From this category we picked a religious website (Dynamic Catholic) and a sexual related chat/forum (LGBTchat.net). Again, the presence of login cookies (i.e., `xf_user` and `frontend_cid`) made the two applications take different execution paths, whose computation time was sufficiently different to be reliably fingerprinted by our solution.

This makes possible for an attacker to differentiate between a user who never accessed the website, one that accessed it, and one that is currently logged in the website. This information can be used for many purposes, such as the one we commented in the case of highly accessed websites, but can also be used for advertisement purposes, as this type of information is not commonly provided by users in their online general use services.

9.4.3 Persistent Login Information

In all previous cases, when the user logs out from the website, the different cookies related to the login status are deleted by the server. In this situation, a third-party website would still be able to detect that the user has visited the website in the past, but it would not be able to distinguish if she had an account and she ever logged into the service.

While this may seem obvious, there are also websites for which it is not true. In fact, some sites do not properly (or at least not completely) delete all cookies they created in relation to the login process. This could be either because the cookie deleting process was not performed correctly, or because the website developers explicitly decided to maintain part of the login information stored in the cookies even when the user is not logged in. However, the presence of these cookies can be sufficient to trigger a different code execution in the application that can be detected by our technique. This allows third-party to be able to differentiate users that just accessed the websites from those who own an account, even if they are not logged in at the time the test is performed.

For instance both Microsoft/MSN and Openload fall into this category

because of the presence of, respectively, a `MUID` cookie and a cookie with a `MD5` hash as name. In both cases, when the user logs out of the service, some cookies are deleted but some other are maintained in the browser. In this two specific cases, we first classified the websites following the same three-state schema as in previous cases. Then, we logged out of the service and checked if this state would be classified as logged or accessed. Our results show that in both cases, the comparison phases classified this state as logged. This indicates that even if the user logged out, if the websites does not correctly delete all related cookies, it would be possible to detect a previous logged state.

9.5 Discussion

Even if a user trusts all the websites she visits, many websites include a large number of third-party services and resources to improve their usage or to monetize their traffic. All these scripts loaded on the website, including simple advertisement banners, can track users and perform many different attacks, including the one we presented in this chapter.

9.5.1 Network Effect

Our attack relies on a time-based side channel used to distinguish among different execution paths on the server-side application code. The fact that BakingTimer does not look at absolute times but at the difference between two consecutive requests minimizes the effects of network delays on our computation. However, even if the time difference between the two request is minimal, it is possible that small non-deterministic variations such as the jitter, bandwidth, or network congestion and routing can introduce a certain level of noise in the measurement. To account for these small fluctuations, BakingTimer needs to repeat each test multiple times. As shown in §9.3, ten comparisons are sufficient to detect nearly 40% of all the analyzed websites, which increases over 50% if we perform a total of 50 comparisons.

9.5.2 Comparison with Similar Techniques

Table 9.3 shows the different state-of-the-art methods and their different characteristics, both in terms of the adopted technique and on the type and size of the experiments performed to validate it.

Table 9.3: A comparison of state-of-the-art methods for history sniffing.

Approach	Conference	Type	Login Status	Difficult Clean	Previous Access	Websites Analyzed
Timing Attacks on Web Privacy [92]	CCS '00	Web & DNS Caching	✗	✗	✓	<10
Exposing private information... [38]	WWW '07	Server Boolean Values	✓	✓	✗	<10
Cross-origin Pixel Stealing [155]	CCS '13	Cross-origin DOM Structure	✓	✓	✗	<10
Identifying Cross-origin... [166]	NDSS '15	HTML5 Application Cache	✓	✗	✗	<10
The Clock is Still Ticking [276]	CCS '15	Cross-origin Resource Size	✓	✓	✗	<10
Our Approach (BakingTimer)	–	Cookie-based Request Timing	✓	✓	✓	10,486

The majority of previous works allowed an attacker to detect the login status of a victim in other websites. Only one [92], apart form ours, allows also to detect the access state (but this same technique is unable to detect the login status). The only technique able to detect both access and login state is the one presented in this chapter.

Most of existing attacks, including our own, do not rely on any other browser resource than the cookies. This makes the technique resilient to generic browsing history cleaning processes, as browsers explicitly discourages user to delete cookies in their settings (see §9.1). Two techniques are instead based on different types of browser caching that, on the contrary of cookies, are even deleted by default, and therefore users can easily and without any major consequence delete them when needed.

Regarding the number of websites analyzed, the difference between this work and any previous one is clear. As nearly all the techniques are only able to detect the login status, the manual effort needed to perform a big scale analysis made large scale experiments unfeasible. We also presented a similar set of experiments in §9.4, but we also performed an automatic analysis of over 10k websites divided in different categories to provide a general overview of how effective this attack can be in the wild.

9.5.3 **Countermeasures**

There are two possible countermeasures that could be implemented in order to avoid different types of server-side timing attacks such as the one presented in this paper [209, 246].

One consists in including a random delay in the response time of the servers. However, this would just increase the noise and a larger number of comparisons may compensate for small random changes in the processing time.

The other method would be to change the web application to have fixed response times for sensitive requests. On top of being very difficult to be properly implemented, this solution would also have a large performance impact — considerably reducing the number of requests per second a web site can sustain. Moreover, as the fixed time would need to be exactly the same for all the sensitive request, they should be as slow as the slowest response the server can make.

For all these reasons, we believe none of the mitigation is practical and feasible to implement in a real-world deployment. Like other time-based network fingerprinting solutions, BakingTimer is therefore very difficult to

mitigate.

9.6 Related Work

History sniffing attacks are a widely explored topic with different techniques and solutions presented over the years. Clover [59] found that it was possible to identify previously visited websites checking the CSS:`visited` style of a specially crafted link though the `getComputedStyle` method in JavaScript. Many other similar attacks appeared using different CSS-based techniques [256, 130, 140, 288]. Kotcher et al. [155] discovered that besides from the above mentioned attacks, the usage of CSS filters allows the involuntary revelation of sensitive data, such as text tokens, exploiting time differences to render various DOM trees. Weinber et al. [283] followed another direction, using interactive techniques to get the information. While these attacks are much slower, the protection methods are in principle more difficult to implement.

With a different approach, and leaving CSS aside, Felten and Schneider [92] introduced web timing attacks as a tool to compromise users private data and, specifically, their web-browsing history. Particularly, they proposed a method based on leveraging the different forms of web browser cache to obtain user specific browsing information. By measuring the time needed to access certain data from a third-party website, the attacker could determine if that specific data was cached or not, indicating a previous access. Some years later, Jia et al. [144] analyzed the possibility of identifying the geo-location of a given visitor using to the customization of services performed by websites. As this location-sensitive content is also cached, it is possible to determine the location by checking this concrete data and without relying in any other technique.

Bortz et al. [38] organized JavaScript web timing attacks in two different types of attacks: (i) direct timing, based on measuring the difference in time of diverse HTTP requests and (ii) cross-site timing, that allows to retrieve private client-side data. The first type could expose data that may be used to prove the validity of specific user information in certain secure website, such as the username. The second attack type follows the same line of previous work by Felten and Schneider. They also performed some experiments that suggested that these timing vulnerabilities were more common than initially expected.

Two recent studies show that these attacks are far from being solved.

Van Goethem et al. [276] proposed new timing techniques based on estimating the size of cross-origin resources. Since the measurement starts after the resources are downloaded, it does not suffer from unfavorable network conditions. The study also shows that these attacks could be used in various platforms, increasing the attack surface and the number of potential victims. The specific size of the resource can leak the current state of the user in the website. Lee et al. [166] demonstrated that using HTML5's AppCache functionality (to enable offline access), an attacker can correctly identify the status of a target URL. This information can later be used to check if a user is logged or not in certain website.

However, these timing techniques can generally only determine if the user is logged on a specific website or some isolated data, but not if she has just previously accessed it. Moreover, some of them use resources easily cleanable by the user, like different cache options, as they do not imply any visible consequence to the final user.

9.7 Conclusions

Many different threats against the users security and privacy can benefit from a list of websites previously accessed by the user and a list of services where the user is logged in or ever logged in.

In this chapter, we show that simply using cookies of third-party websites, is possible the detect the specific state (e.g., accessed and logged) of a user in certain website, which outperforms previous techniques that are only able to detect one single state. In particular, we present a novel timing attack against server-side request processing schema. This technique is capable of detecting execution paths with more than 5 milliseconds of difference between each other.

We also analyzed real-world servers to detect the percentage of websites vulnerable to the presented attack. All previous work analyzed less than 10 websites (manually), as they generally only detect the logged status. We performed this same analysis, and additionally, we performed an automated check of 10k websites from different categories and number of users. Results show that more than half of the websites are vulnerable to our technique.

Part IV

General Summary

"Those who can imagine anything, can create the impossible."

Alan Turing

Chapter

10

Conclusions

We have presented different web security and privacy analyses and new offensive techniques to attack web users with just some lines of JavaScript. This chapter summarizes the result of this research and measures the achievement of the objectives proposed. The remainder of this chapter is organized as follows. Section 10.1 enumerates the main contributions of each of the contributions presented, and Section 10.2 outlines the final remarks.

10.1 Main Contributions

In all previous chapters we have presented the different analyses and new techniques in order to improve the security of web users. Now, we revisit the initial hypothesis defined in the first chapter of this dissertation.

> *"Web security and privacy vulnerations are a worryingly common and known problem in the current Internet. Nevertheless, new offensive techniques and analyses can be performed."*

Next, we summarize the main contributions of this thesis.

Knockin' on Trackers' Door: Large-Scale Automatic Analysis of Web Tracking

- We performed the first large-scale analysis of generic web tracking. The vast majority of the websites performed tracking and more than half of the scripts in them were used to track users online.

- We implemented the first automatic tool to detect generic web tracking using code similarity and machine learning (TrackingInspector). It can identify both known tracking scripts and their modifications, and new potentially unknown tracking scripts.

The Onions Have Eyes: A Comprehensive Structure and Privacy Analysis of Tor Hidden Services

- We performed the largest exploration of the websites hosted on the Tor hidden services. Previous analyses included just to the home pages of each site.

- We present a connectivity and structure analysis, looking at links and redirections, as well as at the external resources imported and exported (both from/to the surface web and other Tor hidden services).

- We also analyzed the privacy implications of Tor hidden services and we measure for the first time the tracking ecosystem in this environment.

Dirty Clicks: a Study of the Usability and Security Implications of Click-related Behaviors on the Web

- We analyzed the click ecosystem by observing the effect nearly 2.5M clicks performed on different websites. Our measurement covers general access domains, such as those in most accessed lists, as well as domains more commonly related to gray activities (e.g., piracy and porn).

- We identified various security and privacy risks that web users might be exposed to while visiting the Web. These dangers range from phishing and man-in-the-middle attacks to cookie control bypasses.

- We show that these threats are real and extremely widespread to the point of affecting the vast majority of the domains we analyzed.
- We propose a list of best practices to guide web developers. We also implemented an online service to automatically check for the presence of bad link-related practices. Our service could be used by web developers to check their webpages as well as by end users to test existing web sites.
- Finally, we propose some countermeasures to be implemented by current browsers to mitigate these problems on the client side and we implemented a browser extension that makes these countermeasures readily available to end users.

Can I Opt Out Yet? GDPR and the Global Illusion of Cookie Control

- We perform a global analysis of how websites handle the new European data protection rules. We check both how the user is informed about cookie tracking, and the actual behavior in terms of tracking cookies installed.
- We discover that the GDPR has a global effect on a very large number of websites, either directly or indirectly. In particular, we found similar behavior in the EU and the USA.
- We discover that, often, opting out is not properly implemented and most websites end up tracking users with long-lasting cookies despite them having opted out.

Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies

- We propose the first time-based extension enumeration attack that can retrieve the complete list of extensions installed in browsers that use access control settings. This method largely outperforms any previous extension fingerprinting methodology presented to date.
- We design a static analysis tool for Safari extensions, and use it to flag hundreds of potentially vulnerable cases in which the developers leaked the random extension URI. Through an exhaustive manual

code analysis on a subset of the extensions, we confirm that this is indeed a very widespread problem affecting a large fraction of all Safari extensions.

- We show that browsers extension resources control policies are very difficult to properly design and implement, and they are prone to subtle errors that undermine their security. Our research led to numerous discussions with the developers of all major browsers and extensions, including the ones vulnerable to our attacks and the ones that are still in the design or testing phase. As a result, our study is helping to secure all browsers against these common errors.

Clock Around the Clock: Time-Based Device Fingerprinting

- We show that a timing side-channel present in all modern computers can be used to uniquely identify a machine among a large number of possible (identical or not) candidates.

- We present a specific implementation of our time-based fingerprinting technique based on simple cryptographic functions. We tested our solution in a homogeneous scenario for fingerprinting evaluations that tackles the main limitations of previous tests by including measures to deal with homogeneous targets.

- We ported our technique to the Web using the HTML5 Cryptography API library. This makes our solution available as a web device fingerprinting technique. We show that our technique overcomes the state-of-the-art hardware-based device fingerprinting techniques both in a homogeneous scenario and in a real-world web fingerprinting experiment.

BakingTimer: Privacy Analysis of Server-Side Request Processing Time

- We present BakingTimer, a new history sniffing technique based on the analysis of the server request processing schema that is able to detect both if the user previously accessed a website or if she is logged in the website. None of the previous techniques was able to detect both things just with one method.

- We analyzed how many websites are vulnerable to the attack, and found that we are able to detect the access to more than half of the websites analyzed. All previous work just tested their techniques with less than 10 websites, we tested over 10K websites from different types and categories, resulting in the largest evaluation of any history sniffing technique to date.

10.2 Final Remarks

In general, web tracking is a widespread technique that despite it can be used to enhance user experience and even its security, it violates user's privacy, sometimes without her explicit knowledge and consent. New techniques have been developed that make fingerprinting and tracking easier and more difficult to combat. In addition, its emerging creation of targeted malware campaigns (as the cases reported by Symantec [249] and FireEye [271] this last year) link directly this technique to the creation of targeted and personalized malicious software, raising a relevant problem not only for user privacy but also in the field of malware analysis that has to cope with more complex samples.

References

[1] Data protection act 2018. http://www.legislation.gov.uk/ukpga/2018/12/contents/enacted.

[2] Data protection in the eu. https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_en.

[3] FreeNet. https://freenetproject.org.

[4] GDPR and cookies. https://www.cookiebot.com/en/gdpr-cookies/.

[5] I2P: The Invisible Internet Project. https://geti2p.net/.

[6] PhantomJS. http://phantomjs.org/.

[7] Tor Project: Anonymity Online. https://www.torproject.org.

[8] Tor2web: Browse the Tor Onion Services. https://www.tor2web.org/.

[9] TorMETRICS. https://metrics.torproject.org.

[10] The GDPR's reach: Material and territorial scope under articles 2 and 3. https://www.wileyrein.com/newsroom-newsletters-item-May_2017_PIF-The_GDPRs_Reach-Material_and_Territorial_Scope_Under_Articles_2_and_3.html, May 2017.

[11] RFC 7235. Section 3.1: 401 unauthorized. https://tools.ietf.org/html/rfc7235/, 2018.

[12] Abine. Tracking list. https://www.abine.com/index.html.

[13] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[14] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: dusting the web for fingerprinters. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[15] AdblockPlus. Easyprivacy. `https://easylist.adblockplus.org/`.

[16] AdGuard. Tracking list. `https://adguard.com/`.

[17] Gaurav Aggarwal, Elie Bursztein, Collin Jackson, and Dan Boneh. An analysis of private browsing modes in modern browsers. In *USENIX Security Symposium*, pages 79–94, 2010.

[18] Istemi Ekin Akkus, Ruichuan Chen, Michaela Hardt, Paul Francis, and Johannes Gehrke. Non-tracking web analytics. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CSS)*, pages 687–698. ACM, 2012.

[19] Ibrahim Altaweel, Jaime Cabrera, Hen Su Choi, Katie Ho, Nathan Good, and Chris Hoofnagle. Web privacy census: Html5 storage takes the spotlight as flash returns.

[20] Apple. Accessing Resources Within Your Extension Folder. `https://developer.apple.com/library/safari/documentation/Tools/Conceptual/SafariExtensionGuide/AccessingResourcesWithinYourExtensionFolder/AccessingResourcesWithinYourExtensionFolder.html`.

[21] Apple. Safari Extensions Development Guide. `https://developer.apple.com/library/safari/documentation/Tools/Conceptual/SafariExtensionGuide`.

[22] Apple. Manage cookies and website data using safari. `https://support.apple.com/kb/ph21411?locale=en_US`, 2018.

[23] Article 29 Data Protection Working Party. Opinion 04/2012 on cookie consent exemption. http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2012/wp194_en.pdf, 2012.

[24] Article 29 Data Protection Working Party. Opinion 9/2014 on the application of directive 2002/58/ec to device fingerprinting. http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp224_en.pdf, 2014.

[25] Atul Varma. Lightbeam: Discover who's tracking you online. http://www.mozilla.org/en-US/lightbeam/.

[26] M Ayenson, DJ Wambach, A Soltani, N Good, and CJ Hoofnagle. Flash cookies and privacy II: Now with HTML5 and Etags respawning (2011). *Social Science Research Network Working Paper Series*, 2011.

[27] Michael Backes, Aniket Kate, Matteo Maffei, and Kim Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2012.

[28] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. A solution for the automated detection of clickjacking attacks. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 135–144. ACM, 2010.

[29] Sruthi Bandhakavi, Nandit Tiku, Wyatt Pittman, Samuel T King, P Madhusudan, and Marianne Winslett. Vetting browser extensions for security vulnerabilities with vex. *Communications of the ACM*, 54(9):91–99, 2011.

[30] Suman Banerjee and Vladimir Brik. Wireless device fingerprinting. In *Encyclopedia of Cryptography and Security*, pages 1388–1390. Springer, 2011.

[31] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting Browsers from Extension Vulnerabilities. In *Pro-*

ceedings of the Network and Distributed Systems Security Symposium (NDSS), 2010.

[32] D Basu. On sampling with and without replacement. *Sankhyā: The Indian Journal of Statistics*, pages 287–294, 1958.

[33] Michael K Bergman. White paper: the deep web: surfacing hidden value. *Journal of electronic publishing*, 7(1), 2001.

[34] Mikhail Bilenko, Matthew Richardson, and Janice Tsai. Targeted, not tracked: Client-side solutions for privacy-friendly behavioral advertising. In *Proceedings of the Privacy Enhancing Technologies (PETS)*, 2011.

[35] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *IEEE Symposium on Security and Privacy (Oakland)*, 2013.

[36] Blocksi. Web content filtering. `http://www.blocksi.net/`.

[37] Duane S Boning and James E Chung. Statistical metrology: Understanding spatial variation in semiconductor manufacturing. In *Proceedings of the Microelectronic Manufacturing*. International Society for Optics and Photonics, 1996.

[38] Andrew Bortz and Dan Boneh. Exposing private information by timing web applications. In *Proceedings of the 16th international conference on World Wide Web*, pages 621–628. ACM, 2007.

[39] Keith A Bowman, Steven G Duvall, and James D Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of solid-state circuits*, 37(2):183–190, 2002.

[40] Harry Brignull. Dark patterns. `https://darkpatterns.org/`.

[41] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Computer networks*, 33(1):309–320, 2000.

[42] Matthew Bryant. and about: to detect firefox & addons. https://thehackerblog.com/dirty-browser-enumeration-tricks-using-chrome-and-about-to-detect-firefox-plugins/index.html.

[43] Ahmet Salih Buyukkayhan, Kaan Onarlioglu, William Robertson, and Engin Kirda. CrossFire: An Analysis of Firefox Extension-Reuse Vulnerabilities. In *Proceedings of the Network and Distributed System Security (NDSS)*, 2016.

[44] Simon Byers, Lorrie Faith Cranor, Dave Kormann, and Patrick McDaniel. Searching for privacy: Design and implementation of a p3p-enabled search engine. In *Privacy Enhancing Technologies*, pages 314–328. Springer, 2005.

[45] Mathias Bynens. About rel=noopener. https://mathiasbynens.github.io/rel-noopener/, 2018.

[46] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th international conference on World Wide Web*, 2011.

[47] Davide Canali, Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. A quantitative study of accuracy in system call-based malware detection. In *Proceedings of the International Symposium on Software Testing and Analysis*, 2012.

[48] Yinzhi Cao, Song Li, and Erik Wijmans. (Cross-)browser fingerprinting via os and hardware level features. In *Proceedings of the Network and Distributed System Symposium (NDSS)*, 2017.

[49] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. An evaluation of the google chrome extension security architecture. In *Proceedings of the USENIX Security Symposium (SEC)*, 2012.

[50] Sambuddho Chakravarty, Georgios Portokalidis, Michalis Polychronakis, and Angelos D Keromytis. Detecting traffic snooping in tor using decoys. In *Proceedings of the International Workshop on Recent Advances in Intrusion Detection (RAID)*, 2011.

[51] Li Chang, Hsu-Chun Hsiao, Wei Jeng, Tiffany Hyun-Jin Kim, and Wei-Hsi Lin. Security implications of redirection trail in popular websites worldwide. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1491–1500. International World Wide Web Conferences Steering Committee, 2017.

[52] Ping Chen, Nick Nikiforakis, Christophe Huygens, and Lieven Desmet. A dangerous mix: Large-scale analysis of mixed-content websites. In *Information Security*, pages 354–363. Springer, 2015.

[53] Chromium. Extension in incognito. https://blog.chromium.org/2010/06/extensions-in-incognito.html.

[54] Ravi Chugh, Jeffrey A Meister, Ranjit Jhala, and Sorin Lerner. Staged information flow for javascript. In *ACM Sigplan Notices*, volume 44, pages 50–62. ACM, 2009.

[55] Vincenzo Ciancaglini, Marco Balduzzi, Max Goncharov, and Robert McArdle. Deepweb and Cybercrime: It's Not All About TOR. http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-cybercrime-and-the-deep-web.pdf, 2013.

[56] Vincenzo Ciancaglini, Marco Balduzzi, Robert McArdle, and Martin Rösler. Below the surface: Exploring the deep web. https://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_below_the_surface.pdf, 2015.

[57] Cliqz. Ghostery. https://www.ghostery.com/.

[58] Cloudacl. Web security service. http://www.cloudacl.com/.

[59] Andrew Clover. Css visited pages disclosure. *BUGTRAQ mailing list posting*, 2002.

[60] comScore. The Impact of Cookie Deletion on Site-Server and Ad-Server Metrics in Australia, 2011.

[61] World Wide Web Consortium. User timing. https://www.w3.org/TR/user-timing/.

[62] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World Wide Web*, 2010.

[63] Charlie Curtsinger, Benjamin Livshits, Benjamin G Zorn, and Christian Seifert. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. In *Proceedings of the USENIX Security Symposium (Sec)*, 2011.

[64] Anupam Das, Nikita Borisov, and Matthew Caesar. Tracking mobile web users through motion sensors: Attacks and defenses. In *Proceedings of the Network and Distributed System Symposium (NDSS)*, 2016.

[65] Nerdy Data. Search engine for source code. `https://nerdydata.com/`.

[66] Vacha Dave, Saikat Guha, and Yin Zhang. Viceroi: Catching click-spam in search ad networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 765–776. ACM, 2013.

[67] Willem De Groef, Dominique Devriese, Nick Nikiforakis, and Frank Piessens. Flowfox: a web browser with flexible and precise information flow control. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 748–759. ACM, 2012.

[68] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. We value your privacy ... now take some cookies: Measuring the GDPR's impact on web privacy, 2018.

[69] Matteo Dell'Amico and Maurizio Filippone. Monte Carlo strength evaluation: Fast and reliable password checking. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 158–169. ACM, 2015.

[70] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password strength: An empirical analysis. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[71] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. Accelprint: Imperfections of accelerometers make smartphones trackable. In *Proceedings of the Network and Distributed System Symposium (NDSS)*, 2014.

[72] Mohan Dhawan and Vinod Ganapathy. Analyzing information flow in JavaScript-based browser extensions. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2009.

[73] Cambridge Dictionary. Cambridge dictionaries online, 2015.

[74] Digital Advertising Alliance. Self-regulatory principles for online behavioral advertising, behavioral advertising. http://www.aboutads.info/resource/download/seven-principles-07-01-09.pdf, 2009.

[75] Digital Advertising Alliance. Self-regulatory principles for multi-site data. http://www.aboutads.info/resource/download/Multi-Site-Data-Principles.pdf, 2011.

[76] Disconnect. Tracking list. https://disconnect.me/.

[77] Vladan Djeric and Ashvin Goel. Securing script-based extensibility in web browsers. In *Proceedings of the USENIX Security Symposium (SEC)*, 2010.

[78] DMA Italia, FedoWEB, Iab Italia, Netcomm, UPA, and Iubenda. Cookies instructions kit. https://help.iubenda.com/wp-content/uploads/2018/04/Cookie-Law-Official-Kit-en.pdf.

[79] Peter Eckersley. How unique is your web browser? In *Proceedings of the Privacy Enhancing Technologies (PETS)*, 2010.

[80] EFF. Privacy Badger. https://www.eff.org/es/node/73969.

[81] Electronic Frontier Foundation. Panopticlick: Is your browser safe against tracking? https://panopticlick.eff.org/.

[82] Steven Englehardt and Arvind Narayanan. Openwpm. https://github.com/citp/OpenWPM.

[83] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the ACM*

SIGSAC Conference on Computer and Communications Security (CCS), 2016.

[84] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the International Conference on World Wide Web*, 2015.

[85] European Parliament. Directive 2002/58/ec. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:en:HTML, 2002.

[86] Directive 2009/136/EC of the European Parliament and of the Council of 25 November 2009 amending Directive 2002/22/EC on universal service and users' rights relating to electronic communications networks and services, Directive 2002/58/EC concerning the processing of personal data and the protection of privacy in the electronic communications sector and Regulation (EC) No 2006/2004 on cooperation between national authorities responsible for the enforcement of consumer protection laws. *Official Journal of the European Union*.

[87] Evidon. Digital governance, privacy compliance, website monitoring. https://www.evidon.com/.

[88] Fanboy. Tracking list. https://www.fanboy.co.nz/.

[89] Federal Trade Commission. Protecting consumer privacy in an era of rapid change, a proposed framework for businesses and policymakers". https://www.ftc.gov/reports/preliminary-ftc-staff-report-protecting-consumer-privacy-era-rapid-change-proposed-framework, 2010.

[90] Federal Trade Commission. Protecting consumer privacy in an era of rapid change: Recommendations for businesses and policymakers". https://www.ftc.gov/reports/protecting-consumer-privacy-era-rapid-change-recommendations-businesses-policymakers, 2012.

[91] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring https adoption on the web. In *26th USENIX Security Symposium*, pages 1323–1338, 2017.

[92] Edward W Felten and Michael A Schneider. Timing attacks on web privacy. In *Proceedings of the 2000 ACM SIGSAC conference on Computer & communications security*, pages 25–32. ACM, 2000.

[93] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656. ACM, 2007.

[94] David Fifield and Serge Egelman. Fingerprinting web users through font metrics. In *International Conference on Financial Cryptography and Data Security*, pages 107–124. Springer, 2015.

[95] FileWatcher. The file search engine. `http://www.filewatcher.com/`.

[96] Russ Fink. A statistical approach to remote physical device fingerprinting. In *Proceedings of the Military Communications Conference (MILCOM)*, 2007.

[97] Ronald A Fisher. *Statistical methods and scientific inference.* Hafner Publishing Co., 1956.

[98] Rudolph Flesch. A new readability yardstick. *Journal of applied psychology*, 32(3):221, 1948.

[99] Florida Statutes. Florida statutes section 627.4145 - readable language in insurance policies. 2016.

[100] Fortinet. Fortiguard web filtering. `http://www.fortiguard.com/`.

[101] Electronic Frontier Foundation. Privacy badger. `https://www.eff.org/es/privacybadger`.

[102] Mozilla Foundation. Public suffix list. `https://publicsuffix.org/list/`.

[103] Mozilla Foundation. Cursor - css: Cascading style sheets. `https://developer.mozilla.org/en-US/docs/Web/CSS/cursor`, 2018.

[104] Mozilla Foundation. Disable third-party cookies in firefox to stop some types of tracking by advertisers. https://support.mozilla.org/en-US/kb/disable-third-party-cookies, 2018.

[105] Mozilla Foundation. Security/Tracking protection. https://wiki.mozilla.org/Security/Tracking_protection, 2018.

[106] Mozilla Foundation. Window.opener. https://developer.mozilla.org/en-US/docs/Web/API/Window/opener, 2018.

[107] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie V Randwyk, and Douglas Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proceedings of the USENIX Security Symposium (SEC)*, 2006.

[108] Yanick Fratantonio, Chenxiong Qian, Simon P Chung, and Wenke Lee. Cloak and dagger: from two permissions to complete control of the ui feedback loop. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 1041–1057. IEEE, 2017.

[109] Matthew Fredrikson and Benjamin Livshits. Repriv: Re-imagining content personalization and in-browser privacy. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2011.

[110] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[111] Sujata Garera, Niels Provos, Monica Chew, and Aviel D Rubin. A framework for detection and measurement of phishing attacks. In *Proceedings of the 2007 ACM workshop on Recurring malcode*, pages 1–8. ACM, 2007.

[112] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the ACM Conference on Computer and CBommunications Security (CCS)*, 2002.

[113] Eric Gerds. Plugindetect. http://www.pinlady.net/PluginDetect/.

[114] GNU/Linux. Stress, tool to impose load on and stress test systems. https://linux.die.net/man/1/stress, 2018.

[115] Avi Goldfarb and Catherine E Tucker. Privacy regulation and online advertising. *Management Science*, 57(1):57–71, 2011.

[116] Nathaniel Good, Rachna Dhamija, Jens Grossklags, David Thaw, Steven Aronowitz, Deirdre Mulligan, and Joseph Konstan. Stopping spyware at the gate: a user study of privacy, notice and spyware. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 43–52. ACM, 2005.

[117] Google. Chrome Web Store. https://www.google.es/chrome/webstore/.

[118] Google. Manifest - web accessible resources. https://developer.chrome.com/extensions/manifest/web_accessible_resources.

[119] Google. What are extensions? https://developer.chrome.com/extensions.

[120] Google. Leak of visited status of page in blink. https://chromereleases.googleblog.com/2018/05/stable-channel-update-for-desktop_58.html, 2018.

[121] Google. Opens external anchors using rel="noopener". https://developers.google.com/web/tools/lighthouse/audits/noopener, 2018.

[122] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Bos, and Cristiano Giuffrida. ASLR on the Line: Practical Cache Attacks on the MMU. In *Proceedings of the Network and Distributed System Symposium (NDSS)*, 2017.

[123] Jens Grossklags and Nathan Good. Empirical studies on software notices to inform policy makers and usability designers. In *International Conference on Financial Cryptography and Data Security*, pages 341–355. Springer, 2007.

[124] WebAssembly W3C Community Group. Webassembly. http://webassembly.org/, 2018.

[125] Arjun Guha, Matthew Fredrikson, Benjamin Livshits, and Nikhil Swamy. Verified security for browser extensions. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2011.

[126] Saikat Guha, Bin Cheng, and Paul Francis. Privad: practical privacy in online advertising. In *Proceedings of the USENIX conference on Networked Systems Design and Implementation (NDSI)*, 2011.

[127] Xiao Han, Nizar Kheir, and Davide Balzarotti. Phisheye: Live monitoring of sandboxed phishing kits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1402–1413. ACM, 2016.

[128] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the deep web. *Communications of the ACM*, 50(5):94–101, 2007.

[129] Daniel Hedin, Arnar Birgisson, Luciano Bello, and Andrei Sabelfeld. Jsflow: Tracking information flow in javascript and its apis. In *Proc. 29th ACM Symposium on Applied Computing*, 2014.

[130] Mario Heiderich, Marcus Niemietz, Felix Schuster, Thorsten Holz, and Jörg Schwenk. Scriptless attacks: stealing the pie without touching the sill. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 760–771. ACM, 2012.

[131] Alex Hern and Jim Waterson. Sites block users, shut down activities and flood inboxes as GDPR rules loom. The Guardian, May 2018.

[132] Ariya Hidayat. Phantomjs. `http://phantomjs.org/`.

[133] Tin Kam Ho. Random decision forests. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1995.

[134] Jun Huang, Wahhab Albazrqaoe, and Guoliang Xing. Blueid: A practical system for bluetooth device identification. In *INFOCOM, 2014 Proceedings IEEE*, pages 2849–2857. IEEE, 2014.

[135] Lin-Shung Huang, Alexander Moshchuk, Helen J Wang, Stuart Schecter, and Collin Jackson. Clickjacking: Attacks and defenses. In *USENIX Security Symposium*, pages 413–428, 2012.

[136] Clint Huffman. *Windows Performance Analysis Field Guide*. Elsevier, 2014.

[137] Intelliagg and Darksum. DEEPLIGHT: shining a light on the dark web. http://www.deep-light.net/, 2016.

[138] Suman Jana and Sneha K Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. *IEEE Transactions on Mobile Computing*, 9(3):449–462, 2010.

[139] Dahlia Janan and David Wray. Readability: The limitations of an approach through formulae. 2012.

[140] Artur Janc and Lukasz Olejnik. Web browser history detection as a real-world privacy threat. In *European Symposium on Research in Computer Security*, pages 215–231. Springer, 2010.

[141] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in javascript web applications. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 270–283. ACM, 2010.

[142] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Bjoern Scheuermann. Anonymously deanonymizing and disabling the tor network. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.

[143] Carlos Jensen and Colin Potts. Privacy policies as decision-making tools: an evaluation of online privacy notices. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 471–478. ACM, 2004.

[144] Yaoqi Jia, Xinshu Dong, Zhenkai Liang, and Prateek Saxena. I know where you've been: Geo-inference attacks via the browser cache. *IEEE Internet Computing*, 19(1):44–53, 2015.

[145] K. Kotowicz. Intro to chrome add-ons hacking. http://blog.otowicz.net/2012/02/intro-to-chrome-addons-hacking.html.

[146] Jeremy Kahn, Stephanie Bodoni, and Stefan Nicola. It'll cost billions for companies to comply with europe's new data law, March 2018.

[147] Samy Kamkar. Evercookie – virtually irrevocable persistent cookies. http://samy.pl/evercookie/.

[148] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *Proceedings of the USENIX Security Symposium (SEC)*, 2014.

[149] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In *USENIX Security Symposium*, pages 637–652, 2013.

[150] Arjaldo Karaj, Sam Macbeth, Rémi Berson, and Josep M. Pujol. WhoTracks.Me: Monitoring the online tracking landscape at scale, 2018.

[151] Kaspersky. Tracking list(abbl). `http://forum.kaspersky.com/`.

[152] J Peter Kincaid, Robert P Fishburne Jr, Richard L Rogers, and Brad S Chissom. Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. 1975.

[153] David Kohlbrenner and Hovav Shacham. Trusted browsers for uncertain times. In *Proceedings of the USENIX Security Symposium (Sec)*, 2016.

[154] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[155] Robert Kotcher, Yutong Pei, Pranjal Jumde, and Collin Jackson. Cross-origin pixel stealing: timing attacks using css filters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1055–1062. ACM, 2013.

[156] Krzysztof Kotowicz and Kyle Osbornand. Advanced chrome extension exploitation. leveraging api powers for better evil. *Black Hat USA*, 2012.

[157] Balachander Krishnamurthy. Privacy leakage on the internet, 2010.

[158] Balachander Krishnamurthy and Craig Wills. Privacy diffusion on the web: a longitudinal perspective. In *Proceedings of the 18th international conference on World Wide Web (WWW)*, 2009.

[159] Balachander Krishnamurthy and Craig E Wills. Generating a privacy footprint on the internet. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 65–70. ACM, 2006.

[160] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *Proceedings of the USENIX Security Symposium (SEC)*, 2015.

[161] Fabian Lanze, Andriy Panchenko, Benjamin Braatz, and Thomas Engel. Letting the puss in boots sweat: Detecting fake access points using dependency of clock skews on temperature. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 3–14. ACM, 2014.

[162] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2016.

[163] Issie Lapowsky. California unanimously passes historic privacy bill. Wired, 06 2018.

[164] Dave Lee. Web software firm taunts UK data regulator over cookies. `https://www.bbc.co.uk/news/technology-19505835`, 9 2012.

[165] Jae W Lee, Daihyun Lim, Blaise Gassend, G Edward Suh, Marten Van Dijk, and Srinivas Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *Proceedings of the Symposium on VLSI Circuits*. IEEE.

[166] Sangho Lee, Hyungsub Kim, and Jong Kim. Identifying cross-origin resource status using application cache. In *NDSS*, 2015.

[167] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. In *Proceedings of the USENIX Security Symposium (SEC)*, 2016.

[168] Sarah Jamie Lewis. Onionscan report: April 2016. `https://onionscan.org/reports/april2016.html/`, 2016.

[169] Sarah Jamie Lewis. Onionscan report: July 2016 - https somewhere sometimes. https://mascherari.press/onionscan-report-july-2016-https-somewhere-sometimes/, 2016.

[170] Sarah Jamie Lewis. Onionscan report: June 2016. https://mascherari.press/onionscan-report-june-2016/, 2016.

[171] Sarah Jamie Lewis. Onionscan report: May 2016. https://onionscan.org/reports/may2016.html, 2016.

[172] Timothy Libert. An automated approach to auditing disclosure of third-party data collection in website privacy policies. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 207–216. International World Wide Web Conferences Steering Committee, 2018.

[173] Einar Lielmanis. Js beautifier. http://jsbeautifier.org/.

[174] Lei Liu, Xinwen Zhang, Guanhua Yan, and Songqing Chen. Chrome Extensions: Threat Analysis and Countermeasures. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2012.

[175] Wei Liu, Xiaofeng Meng, and Weiyi Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22(3):447–460, 2010.

[176] Long Lu, Roberto Perdisci, and Wenke Lee. Surf: detecting and measuring search poisoning. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 467–476. ACM, 2011.

[177] Christian Ludl, Sean McAllister, Engin Kirda, and Christopher Kruegel. On the effectiveness of techniques to detect phishing sites. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 20–39. Springer, 2007.

[178] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317, 1957.

[179] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *Proceedings of the VLDB Endowment*, 1(2):1241–1252, 2008.

[180] MalwareBytes. Unusual exploit kit targets chinese users (part 1). https://blog.malwarebytes.org/exploits-2/2015/05/unusual-exploit-kit-targets-chinese-users-part-1/.

[181] MalwareBytes. Unusual exploit kit targets chinese users (part 2). https://blog.malwarebytes.org/intelligence/2015/06/unusual-exploit-kit-targets-chinese-users-part-2/.

[182] Robert Martin, John Demme, and Simha Sethumadhavan. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2012.

[183] Jonathan Mayer. Tracking the trackers: early results. http://cyberlaw.stanford.edu/blog/2011/07/tracking-trackers-early-results, 2011.

[184] Jonathan Mayer and Arvind Narayanan. Do not track. universal web tracking opt out. http://donottrack.us, 2011.

[185] Jonathan R Mayer. Any person... a pamphleteer": Internet anonymity in the age of web 2.0, 2009. Senior Thesis, Stanford University.

[186] Jonathan R Mayer and John C Mitchell. Third-party web tracking: Policy and technology. In *Proceedings of the IEEE International Symposium on Security and Privacy (Oakland)*, 2012.

[187] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the tor network. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[188] Aleecia M Mcdonald, Robert W Reeder, Patrick Gage Kelley, and Lorrie Faith Cranor. A comparative study of online privacy policies and formats. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 37–55. Springer, 2009.

[189] MeanPath. The source code search engine. https://meanpath.com/.

[190] William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *USENIX Security Symposium*, pages 175–191, 2016.

[191] Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. Graph structure in the web-revisited. In *Proceedings of the International World Wide Web Conference (WWW)*, 2014.

[192] Brad Miller, Paul Pearce, Chris Grier, Christian Kreibich, and Vern Paxson. What's clicking what? techniques and innovations of today's clickbots. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 164–183. Springer, 2011.

[193] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2011.

[194] Anthony D Miyazaki. Online privacy and the disclosure of cookie use: Effects on consumer trust and anticipated patronage. *Journal of Public Policy & Marketing*, 27(1):19–33, 2008.

[195] Rani Molla. Advertisers will spend $40 billion more on internet ads than on tv ads this year. Recide, 2018.

[196] Lou Montulli and David M. Kristol. HTTP State Management Mechanism. RFC 2965, October 2000.

[197] Tyler Moore and Benjamin Edelman. Measuring the perpetrators and funders of typosquatting. In *International Conference on Financial Cryptography and Data Security*, pages 175–191. Springer, 2010.

[198] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting information in javascript implementations. In *Proceedings of the Web 2.0 Workshop on Security and Privacy (W2SP)*, 2011.

[199] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of the Web 2.0 Workshop on Security and Privacy (W2SP)*, 2012.

[200] Mozilla. Add-ons for Firefox. `https://addons.mozilla.org/es/firefox/`.

[201] Mozilla. Chrome registration. https://developer.mozilla.org/en-US/docs/Chrome_Registration.

[202] Mozilla. JetPack Project. https://wiki.mozilla.org/Jetpack.

[203] Mozilla. Privacy and the :visited selector. https://developer.mozilla.org/en-US/docs/Web/CSS/Privacy_and_the_:visited_selector.

[204] Mozilla. WebExtension Add-ons. https://developer.mozilla.org/en-US/Add-ons/WebExtensions.

[205] Mozilla. XPCOM Reference. https://developer.mozilla.org/en/docs/Mozilla/Tech/XPCOM/Reference.

[206] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, Edgar Weippl, and FC Wien. Fast and reliable browser identification with javascript engine fingerprinting. In *Proceedings of the Web 2.0 Workshop on Security and Privacy (W2SP)*, 2013.

[207] Steven J Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 27–36. ACM, 2006.

[208] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2005.

[209] Yoshitaka Nagami, Daisuke Miyamoto, Hiroaki Hazeyama, and Youki Kadobayashi. An independent evaluation of web timing attack and its countermeasure. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 1319–1324. IEEE, 2008.

[210] NAI Consumer. Opt out of interest-based advertisement. http://optout.networkadvertising.org.

[211] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 173–187. IEEE, 2009.

[212] Sani R Nassif. Modeling and forecasting of manufacturing variations. In *Proceedings of the International Workshop on Statistical Metrology*, 2000.

[213] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[214] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the International Conference on World Wide Web (WWW)*.

[215] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of IEEE Symposium on Security and Privacy (Oakland)*, 2013.

[216] Web of Trust. Crowdsourced web safety. `https://www.mywot.com/`.

[217] Alessandro Oltramari, Dhivya Piraviperumal, Florian Schaub, Shomir Wilson, Sushain Cherivirala, Thomas B Norton, N Cameron Russell, Peter Story, Joel Reidenberg, and Norman Sadeh. PrivOnto: A semantic framework for the analysis of privacy policies. *Semantic Web*, (Preprint):1–19, 2017.

[218] Kaan Onarlioglu, Mustafa Battal, William Robertson, and Engin Kirda. Securing legacy firefox extensions with SENTINEL. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2013.

[219] Kaan Onarlioglu, Ahmet Salih Buyukkayhan, William Robertson, and Engin Kirda. Sentinel: Securing legacy firefox extensions. *Computers & Security*, 49:147–161, 2015.

[220] OneTrust. Privacy management software. `https://www.onetrust.com/`.

[221] OWASP. Reverse tabnabbing. `https://www.owasp.org/index.php/Reverse_Tabnabbing`, 2018.

[222] Xiang Pan, Yinzhi Cao, and Yan Chen. I Do Not Know What You Visited Last Summer: Protecting Users from Third-party Web Tracking with TrackingFree Browser. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, 2015.

[223] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

[224] European Parliment. Regulation (eu) 2016/679 of 27 april 2016. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679.

[225] Paul Pearce, Vacha Dave, Chris Grier, Kirill Levchenko, Saikat Guha, Damon McCoy, Vern Paxson, Stefan Savage, and Geoffrey M Voelker. Characterizing large-scale click fraud in zeroaccess. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 141–152. ACM, 2014.

[226] Piwik. Turn on/off GDPR compliance on the website. https://help.piwik.pro/consent-manager/setting-consent-manager/.

[227] Libor Polčák and Barbora Franková. On reliability of clock-skew-based remote computer identification. In *Security and Cryptography (SECRYPT), 2014 11th International Conference on*, pages 1–8. IEEE, 2014.

[228] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu, et al. The ghost in the browser: Analysis of web-based malware. *HotBots*, 7:4–4, 2007.

[229] Quantcast. AI-driven audience insights, targeting & measurement. https://www.quantcast.com/.

[230] M Zubair Rafique, Tom Van Goethem, Wouter Joosen, Christophe Huygens, and Nick Nikiforakis. It's free for a reason: Exploring the ecosystem of free live streaming services. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[231] Moheeb Abu Rajab, Lucas Ballard, Panayiotis Mavrommatis, Niels Provos, and Xin Zhao. The nocebo effect on the web: An analysis of fake anti-virus distribution. In *LEET*, 2010.

[232] Aza Raskin. Tabnabbing: A new type of phishing attack. http://www.azarask.in/blog/post/a-new-type-of-phishing-attack/, 2018.

[233] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

[234] Konrad Rieck, Tammo Krueger, and Andreas Dewald. Cujo: efficient detection and prevention of drive-by-download attacks. In *Proceedings of the Annual Computer Security Applications Conference (CSS)*, 2010.

[235] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 12–12. USENIX Association, 2012.

[236] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. *IEEE Oakland Web*, 2(6), 2010.

[237] Timothy J Salo. Multi-factor fingerprints for personal computer hardware. In *Proceedings of the Military Communications Conference (MILCOM)*. IEEE, 2007.

[238] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[239] Amirali Sanatinia and Guevara Noubir. Onionbots: Subverting privacy infrastructure for cyber attacks. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2015.

[240] Amirali Sanatinia and Guevara Noubir. Honey onions: a framework for characterizing and identifying misbehaving tor hsdirs. In *Proceedings of IEEE Conference on Communication Networks Security*, 2016.

[241] Iskander Sanchez-Rola, Davide Balzarotti, and Igor Santos. The Onions Have Eyes: A Comprehensive Structure and Privacy Analysis

of Tor Hidden Services. In *Proceedings of the International World Wide Web Conference (WWW)*, WWW 17, Perth, Australia, April 2017.

[242] Iskander Sanchez-Rola and Igor Santos. Known and Unknown Generic Web Tracking Analyzer: A 1 Million Website Study. Technical report, DeustoTech, University of Deusto, 2016.

[243] Iskander Sanchez-Rola and Igor Santos. Knockin on trackers door: Large-scale automatic analysis of web tracking. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pages 281–302. Springer, 2018.

[244] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In *Proceedings of the USENIX Security Symposium (Sec)*, 2017.

[245] Iskander Sánchez-Rola, Xabier Ugarte-Pedrero, Igor Santos, and Pablo G Bringas. Tracking users like there is no tomorrow: Privacy on the current internet. In *Proceedings of the International Conference on Computational Intelligence in Security for Information Systems (CISIS)*. Springer, 2015.

[246] Sebastian Schinzel. An efficient mitigation method for timing side channels on the web. In *2nd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2011.

[247] John Schwartz. Giving the web a memory cost its users privacy. `http://www.nytimes.com/2001/09/04/technology/04COOK.html`, 2001.

[248] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript . In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, 2017.

[249] Security Response, Symantec. The Waterbug attack group. `http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/waterbug-attack-group.pdf`, 2015.

[250] Alexander Selzer. Beaverbird. https://github.com/AlexanderSelzer/BeaverBird.

[251] Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, and Simon Gibbs. Jalangi: A selective record-replay and dynamic analysis framework for javascript. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 488–498. ACM, 2013.

[252] Amazon Web Services. Alexa top sites. https://aws.amazon.com/es/alexa-top-sites/.

[253] Sergey Shekyan, Ben Vinegar, and Bei Zhang. Phantomjs hide and seek. https://github.com/ikarienator/phantomjs_hide_and_seek.

[254] Natasha Singer. Do not track? advertisers say "don't tread on us". http://www.nytimes.com/2012/10/14/technology/do-not-track-movement-is-drawing-advertisers-fire.html, 2012.

[255] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. The cracked cookie jar: Http cookie hijacking and the exposure of private information. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 724–742. IEEE, 2016.

[256] Michael Smith, Craig Disselkoen, Shravan Narayan, Fraser Brown, and Deian Stefan. Browser history re: visited. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. USENIX Association, 2018.

[257] Ashkan Soltani, Shannon Canty, Quentin Mayo, Lauren Thomas, and Chris Jay Hoofnagle. Flash cookies and privacy. In *Proceedings of the AAAI Spring Symposium: Intelligent Information Privacy Management*, volume 2010, 2010.

[258] Kyle Soska and Nicolas Christin. Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *Proceedings of USENIX Security Symposium (SEC)*, 2015.

[259] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[260] Oleksii Starov and Nick Nikiforakis. Xhound: Quantifying the fingerprintability of browser extensions. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2017.

[261] Deian Stefan, Pablo Buiras, Edward Z Yang, Amit Levy, David Terei, Alejandro Russo, and David Mazières. Eliminating cache-based timing attacks with instruction-based scheduling. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. Springer, 2013.

[262] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 133–144. ACM, 2013.

[263] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: routing attacks on privacy in Tor. In *Proceedings of the USENIX Security Symposium (SEC)*, 2015.

[264] Web Technology Surveys. Usage of javascript websites. `https://w3techs.com/technologies/`.

[265] Michael Sutton. A wolf in sheep's clothing, the dangers of persistent web browser storage. *Black Hat DC 2009 Briefings Speakers*, 2009.

[266] Aaron Swartz. Web.py web framework. `http://webpy.org/`.

[267] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich. The long" taile" of typosquatting domain names. In *USENIX Security Symposium*, pages 191–206, 2014.

[268] AVG Technologies. Privacyfix tracking list. `http://privacyfix.com`.

[269] Mike Ter Louw, Jin Soon Lim, and VN Venkatakrishnan. Extensible web browser security. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2007.

[270] Mike Ter Louw, Jin Soon Lim, and VN Venkatakrishnan. Enhancing web browser security against malware extensions. *Journal in Computer Virology*, 4(3):179–195, 2008.

[271] Threat Intelligence, FireEye. Pinpointing Targets: Exploiting Web Analytics to Ensnare Victims. https://www2.fireeye.com/rs/848-DID-242/images/rpt-witchcoven.pdf, 2015.

[272] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings of the Network and Distributed System Symposium (NDSS)*, 2010.

[273] TrustArc. Truste tracking list. https://www.truste.com/.

[274] TRUSTe. Your advertising choices. http://preferences-mgr.truste.com/.

[275] Rob van Eijk. Tracking detection system (tds). https://github.com/rvaneijk/ruleset-for-AdBlock.

[276] Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. The clock is still ticking: Timing attacks in the modern web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1382–1393. ACM, 2015.

[277] Valentin Vasilyev. Fingerprintjs. https://github.com/Valve/fingerprintjs.

[278] Antoine vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2018.

[279] W3C. Web Cryptography API. https://w3c.github.io/webcrypto/Overview.html, 2018.

[280] Lei Wang, Ji Xiang, Jiwu Jing, and Lingchen Zhang. Towards fine-grained access control on browser extensions. In *Proceedings of the International Conference on Information Security Practice and Experience*, 2012.

[281] Yinglei Wang, Wing-kei Yu, Shuo Wu, Greg Malysa, G Edward Suh, and Edwin C Kan. Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2012.

[282] Webutation. Open website reputation. http://www.webutation.net/.

[283] Z Weinber, E Chen, PR Jayaraman, and C Jackson. I still know what you visited last summer. In *IEEE Symposium on Security and Privacy. Oakland, CA*, pages 20–25, 2011.

[284] William West and S Monisha Pulimood. Analysis of privacy and security in html5 web storage. *Journal of Computing Sciences in Colleges*, 27(3):80–87, 2012.

[285] Daniel Lowe Wheeler. zxcvbn: Low-budget password strength estimation. In *USENIX Security Symposium*, pages 157–173, 2016.

[286] Whotracks.me. GDPR—what happened?, 2018.

[287] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled onions: Exposing malicious tor exit relays. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2014.

[288] Gilbert Wondracek, Thorsten Holz, Engin Kirda, and Christopher Kruegel. A practical attack to de-anonymize social network users. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 223–238. IEEE, 2010.

[289] Gilbert Wondracek, Thorsten Holz, Christian Platzer, Engin Kirda, and Christopher Kruegel. Is the internet for porn? an insight into the online adult industry. In *WEIS*, 2010.

[290] World Wide Web Consortium. Platform for privacy preferences (p3p) project. http://www.w3.org/P3P.

[291] Haidong Xia and José Carlos Brustoloni. Hardening web browsers against man-in-the-middle and eavesdropping attacks. In *Proceedings of the 14th international conference on World Wide Web*, pages 489–498. ACM, 2005.

[292] Fabian Yamaguchi, Felix Lindner, and Konrad Rieck. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning. In *Proceedings of the USENIX Conference on Offensive Technologies (WOOT)*, 2011.

[293] Your Online Choice. A guide to online behavioural advertisement. http://www.youronlinechoices.com/es/preferencias/.

[294] YourAdChoices. WebChoices: Digital advertising alliance's consumer choice tool. http://optout.aboutads.info.

[295] Yue Zhang, Jason I Hong, and Lorrie F Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th international conference on World Wide Web*, pages 639–648. ACM, 2007.

[296] Bing Zhou, Yiyu Yao, and Jigang Luo. Cost-sensitive three-way email spam filtering. *Journal of Intelligent Information Systems*, 42(1):19–45, 2014.

[297] Sebastian Zimmeck and Steven M. Bellovin. Privee: An architecture for automatically analyzing web privacy policies. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 1–16, San Diego, CA, 2014. USENIX Association.

This work is partially supported by the Basque Government
under a pre-doctoral grant given to Iskander Sanchez-Rola.

This dissertation was finished
in Bilbao on September 2018 .